Fachhochschule Köln
Cologne University of Applied Sciences

Kriterien für die Abbildung von XML-Strukturen auf relationale Datenbanken bezogen auf ein Redaktionssystem

Bachelorarbeit

ausgearbeitet von

Christopher Messner

zur Erlangung des akademischen Grades

Bachelor of Science

vorgelegt an der

Fachhochschule Köln
Campus Gummersbach
Fakultät für Informatik und
Ingenieurwissenschaften

im Studiengang

Medieninformatik

Erster Prüfer: Prof. Dr. Kristian Fischer
Fachhochschule Köln

Zweiter Prüfer: Karl Jost
CBC Cologne Broadcasting Center GmbH

Gummersbach, im März 2012

Adressen: Christopher Messner
Nobelstr. 22
51643 Gummersbach
christopher.messner@gmx.de

Prof. Dr. Kristian Fischer
Fachhochschule Köln
Institut für Informatik
Steinmüllerallee 1
51643 Gummersbach
kristian.fischer@fh-koeln.de

Karl Jost
CBC Cologne Broadcasting Center GmbH
Picassoplatz 1
50679 Köln
karl.jost@cbc.de

Kurzfassung

Für Anwendungen, die mit XML-Dokumenten arbeiten, kann es aus der Sicht eines Unternehmens sinnvoll sein, die Daten in einer relationalen Datenbank zu speichern. Sollte die Anwendung von sich aus nur mit nativen XML Datenbanken arbeiten und gar keine Anbindung an relationale Datenbanken vorsehen, sind umfassende Untersuchungen notwendig. Vor allem bei Redaktionssystemen, die für den Produktiv-Betrieb eines Unternehmens essentiell sind, darf es zu keinen Beeinträchtigungen durch die Datenhaltung kommen. Redaktionssysteme arbeiten zudem aufgrund ihrer Funktionsvielfalt mit verschieden strukturierten Dokumenten, an die unterschiedliche Anforderungen bestehen. Es müssen also optimale Strukturen für die einzelnen Dokument-Gruppen gefunden werden. Möglichkeiten XML-Dokumente in einer relationalen Datenbank zu speichern gibt es allerdings viele. Sei es ein nativer XML-Datentyp, ein Mapping auf Relationen oder gar ein hybrides Verfahren. Die Wahl der passenden Struktur wird nicht nur durch die Anwendung, die Nutzer und die Datenbanken beeinflusst, sondern auch durch die Struktur und die Nutzungskontexte der XML-Dokumente. Ziel dieser Arbeit ist es, anhand von ermittelten Kriterien, optimale Strukturen für verschiedenartige XML-Dokumente zu finden. Dazu werden neben dem Redaktionssystem, den Datenbanken und den Daten, auch verschiedene Strukturen untersucht. Abgesehen von den Strukturlösungen sollen die Erkenntnisse in dieser Arbeit dabei helfen, die native XML Datenbank des betroffenen Redaktionssystems durch eine relationale Datenbank zu ersetzen.

Inhaltsverzeichnis

1 Einleitung und Vorgehensweise

1.1 Problemstellung

Gerade in der Zeit, in der XML immer populärer und als Datenaustauschformat interessanter wird, gibt es zahlreiche Untersuchungen, die sich mit der Abbildung von XML-Strukturen auf relationale Datenbanken beschäftigen. Die relationalen Datenbankhersteller bieten mittlerweile auch einen nativen XML-Datentyp an. Man könnte die XML-Dokumente aber auch fragmentiert in verschiedenen Granulierungsgraden speichern. Sollte eine native XML-Datenbank nicht das Mittel der Wahl sein, so muss eine passende Struktur für die Daten gefunden werden. Die Wahl der richtigen Struktur hängt dabei natürlich von der Anwendung ab. Gerade wenn viele Nutzer mit vielen Daten gleichzeitig arbeiten, ist die Datenhaltung besonders wichtig. Redaktionssysteme sind ein passendes Beispiel für solche Anwendungen. Es kommen andauernd aktuelle Agenturmeldungen aus aller Welt in das System und dienen Redakteuren zur Recherche. Zusätzlich arbeiten mehrere Benutzer an Sendeplänen (zum Teil simultan), die letztendlich freigegeben werden und auf Sendung gehen. Dabei spielen unter anderem die Performance und Datenbankeigenschaften wie *concurrency* eine wichtige Rolle.

1.2 Aufgabenstellung

Die Untersuchungen basieren auf einem Projekt (mehr dazu in Kapitel 2), welches bei CBC - Cologne Broadcasting Center GmbH in Köln stattfand. Dort ist das betrachtete Redaktionssystem zum Beispiel für die Fernsehsender RTL und N-TV im Einsatz.
Das Redaktionssystem *NCPower* (mehr dazu in Kapitel 3) arbeitet mit der nativen XML Datenbank Tamino XML Server 4.4.1. Diese soll durch die relationale Datenbank SQL Server 2008 von Microsoft ersetzt werden. Die Gründe dafür sind unter anderem die verbesserten Administrations-, Datensicherungs- und Monitoring-Möglichkeiten bei SQL Server 2008. In dem vorangegangenen Projekt wurde die Übersetzbarkeit der Abfragesprache von Tamino (X-Query) in das entsprechende Microsoft Pendant untersucht und nachgewiesen. Weiterführend wurde ein Gateway-Prototyp zwischen *NCPower* und SQL Server 2008 entwickelt, welcher die Übersetzung in beide Richtungen übernimmt. Das Gateway nimmt also Querys für Tamino entgegen und übersetzt diese in ein SQL

Statements, welche an SQL Server 2008 geschickt werden. Das Ergebnis wird um einen taminospezifischen Wrapper erweitert und an *NCPower* in Form eines XML-Dokuments zurückgeschickt.
Nun stellt sich die Frage, wie die XML-Strukturen von Tamino am Besten in SQL Server 2008 abgebildet werden. Die verschiedenartigen XML-Dokumente sind in Collections (passend zu den *NCPower*-Services) gesammelt. An die Collections bestehen unterschiedliche Anforderungen, da zum Beispiel mit einem Sendeplan anders gearbeitet wird als mit aktuellen Agentur-Meldungen. Wichtig ist auch, dass sich die Struktur der XML-Dokumente gegebenenfalls auf die Operatoren auswirkt, welche auf die Daten angewendet werden. Es gilt also herauszufinden welche Struktur der Daten sich für welches Einsatzgebiet (Collection / Service) am besten eignet. Dazu müssen neben dem Redaktionssystem, den Datenbanken und den Daten auch verschiedene Strukturen untersucht werden.

1.3 Zielsetzung

Nach einer kurzen Betrachtung des vorangegangenen Projekts (Kapitel 2), auf dem diese Arbeit aufbaut, folgen die Untersuchungen der betroffenen Systeme. Zuerst muss das Redaktionssystem *NCPower* (Kapitel 3) betrachtet werden, damit die Anforderungen der Nutzer an das System (und dahingehend die Anforderungen an die Datenbank) festgestellt werden können. Neben dem Redaktionssystem werden die Datenbanken (Kapitel 4) Tamino XML Server 4.4.1 und Microsoft SQL Server 2008 genauer untersucht. Anhand dieser Ergebnisse wird deutlich, welche Voraussetzungen und Einschränkungen durch die Datenbanken gegeben sind. Zusätzlich müssen die einzelnen Tamino-Collections genauer betrachtet werden, da mit den XML-Dokumenten je nach Collection unterschiedlich gearbeitet wird. Die Anforderungen an die Datenhaltung variieren also je nach Collection. Daraufhin werden verschiedene Strukturalternativen und Mapping-Verfahren analysiert (Kapitel 5), um deren Vor- und Nachteile für bestimmte Anwendungsgebiete festzustellen. Die Erkenntnisse aus der Strukturanalyse werden anhand verschiedener Tests evaluiert (Kapitel 6). Dadurch zeigt sich, inwiefern sich die Strukturen für die betroffenen Anwendungsgebiete eignen. Aus diesen Ergebnissen können dann Strukturlösungen für die einzelnen Collections erarbeitet werden (Kapitel 7). Diese Arbeit wird kein allgemeingültiges Lösungsverfahren für jegliche Anwendungen liefern. Allerdings können die Kriterien für die Strukturauswahl (Kapitel 7.2) auch für andere Anwendungen adaptiert werden.

2 Grundlagen und Vorbedingungen

Die Untersuchungen dieser Arbeit bauen auf einem Projekt[1] auf, in welchem die Übersetzbarkeit zweier XQuery Dialekte untersucht und ein dazugehöriger Übersetzer entwickelt wurde. Das Projekt fand bei CBC Cologne Broadcasting Center GmbH in Köln im Ressort Datenbanken statt. CBC gehört, wie auch unter anderem die Sender RTL und N-TV, zu der Mediengruppe RTL Deutschland. Im Unternehmen ist das Redaktionssystem *NCPower* im Einsatz. Mit *NCPower* können Redakteure in eingehenden Agenturmeldungen recherchieren und Beiträge erstellen. Hauptsächlich werden mit dem Redaktionssystem aber die Sendepläne erstellt und verwaltet. Diese gehen direkt von *NCPower* aus auf Sendung. Mit den Sendeplänen können Beiträge verknüpft oder auch direkt Teleprompter-Texte geschrieben werden. Teilweise wird natürlich auch gleichzeitig an ein und demselben Sendeplan gearbeitet. Dies hat Auswirkungen auf die Datenhaltung und die Operatoren mit denen auf die Daten zugegriffen wird. Ein reibungsloses und performantes Arbeiten muss für die Nutzer unbedingt gewährleistet sein.
NCPower arbeitet mit der nativen XML-Datenbank Tamino XML Server 4.4.1 (im Folgenden Tamino). Diese soll durch die relationale Datenbank Microsoft SQL Server 2008 ersetzt werden. Gründe dafür sind unter anderem, dass im Unternehmen die Datenbank Microsoft SQL Server 2008 am verbreitetsten ist und die Administrations-, Datensicherungs- und Monitoring-Möglichkeiten bei dieser weitaus besser sind als bei Tamino. Mit dem Datenbankwechsel würde die einzige nicht relationale Datenbank im Unternehmen abgesetzt werden. Das Problem an der Sache ist, dass *NCPower* keine Anbindung an Microsoft SQL Server 2008 vorsieht.
Eine mögliche Problemlösung ist die Entwicklung eines Gateways zwischen *NCPower* und Microsoft SQL Server 2008. Das Gateway soll die Anfragen von *NCPower* an Tamino entgegen nehmen und die übersetzte Query an den SQL Server 2008 schicken. Das Ergebnis muss noch um einen taminospezifischen Wrapper erweitert werden, welcher entsprechende Statusmeldungen beinhaltet. Die Übersetzung erfolgt also in beide Richtungen. Das Ergebnis wird dann an *NCPower* als application/xml MIME-Type zurück geschickt. Wichtig ist, dass das Gateway performant und stabil läuft. Natür-

[1] Die Unterlagen dazu befinden sich auf einer beiliegenden CD. Sie sind für die Performance-Untersuchung beziehungsweise die Tamino-Semantik der Test-Querys wichtig.

lich müssen auch alle Funktionen von Tamino in einer entsprechenden Form abgebildet werden. Bei diesen Funktionen handelt es sich allerdings nur um solche, die über die APIs aufgerufen werden können.
Es gäbe natürlich auch noch weitere Möglichkeiten das Problem zu lösen. Da der Quellcode von *NCPower* zugänglich ist, würde sich auch eine Applikations-Erweiterung anbieten. So würde die Kommunikation mit SQL Server 2008 direkt in der Anwendung verankert sein. Nichtsdestotrotz muss für eine optimale Speicherung der Tamino XML-Dokumente in SQL Server gesorgt sein.
Für die Gateway-Lösung wurde in dem Projekt also untersucht, ob die XQuery Dialekte von Tamino überhaupt in ein entsprechendes SQL Pendant übersetzt werden können. Nachdem dies erfolgreich nachgewiesen wurde, wurde als Proof-of-Concepts ein Übersetzer entwickelt, welcher Tamino X-Query Abfragen in entsprechende SQL Querys übersetzt. Das Ergebnis wurde mit identischen Testdatensätzen (in beiden Datenbanken) und Querys aus dem Produktivbetrieb von *NCPower* evaluiert. Der Übersetzer wurde schließlich zu einem Gateway-Prototypen mit zusätzlichen Funktionen weiterentwickelt. Es ist bereits möglich Querys, Inserts, Updates, Deletes und administrative Kommandos per http POST an das Gateway zu schicken. Die entsprechenden Informationen werden aus den Parametern des *multipart form* ausgelesen und verarbeitet. Der Wrapper, um den das Ergebnis erweitert wird, enthält zum Beispiel Informationen über eine mögliche Cursorposition oder aber Fehlercodes. Der Gateway-Prototyp unterstützt noch keine Cursor, Transaktionen und Sessions (also Mehrbenutzerbetrieb), was für eine Inbetriebnahme allerdings essentiell ist.
Zu Testzwecken wurde in SQL Server 2008 eine Tamino Collection als eine relationale Tabelle abgebildet. Diese Tabelle bestand aus einer Index-Spalte und einer Spalte vom XML-Datentyp. Für die Tests wurde jedes XML-Dokument aus der Collection in einer eigenen Zeile gespeichert. Dabei hat sich herausgestellt, dass ein Mapping der XML-Dokumente auf relationale Tabellen sinnvoll sein könnte. Dies hängt aber unter anderem von der Collection an sich und der Art der Querys ab. Wichtig ist vor allem, dass die Anforderungen an die Collections performant von der neuen Struktur erfüllt werden können.

3 Untersuchung des Redaktionssystems

Das Redaktionssystem, welches im Zusammenhang mit der Abbildung von XML–Strukturen auf relationale Datenbanken untersucht wurde, ist *NCPower* von NorCom[1]. *NCPower* bietet nicht nur Zugang zu aktuellen Agenturmeldungen, es erlaubt unter anderem auch das Verfassen von Redaktionsbeiträgen, sowie die Planung und Überwachung des Sendeablaufs. In den folgenden Abschnitten wird dieses Redaktionssystem untersucht und analysiert. Dadurch können die Anforderungen an das Redaktionssystem und damit an die Datenbank und die einzelnen Collections ermittelt werden.
Vorab ist anzumerken, dass *NCPower* nicht immer von NorCom entwickelt wurde. NorCom hat die Firma MaxiMedia Technologies GmbH übernommen, welche das Redaktionssystem *Mpower* entwickelt hat. Nach der Übernahme wurde *Mpower* dann in *NCPower* umbenannt. Aus diesem Grund sind die Dokumentationen über das Redaktionssystem, die im Rahmen dieser Arbeit verwendet wurden, von MaxiMedia und nicht von NorCom. In diesen Dokumentationen wird deshalb auf *Mpower* und nicht *NCPower* Bezug genommen. Aktuelle Versionen der Unterlagen standen nicht zur Verfügung.

3.1 Funktion und Aufbau von NCPower

Zuallererst wird der Funktionsumfang, der Aufbau und die Architektur von *NCPower* genauer untersucht. Daraus kann auf die Anforderungen des Redaktionssystems an die Datenbank geschlossen werden. Das Haupteinsatzgebiet von *NCPower* ist die Sendeplanerstellung und -verwaltung. Dazu ist anzumerken, dass je nach Nutzerberechtigungen der Funktionsumfang mehr oder weniger eingeschränkt ist. Der Sendeplan beziehungsweise einzelne Skripte im Plan können zum Beispiel nicht von jedermann freigegeben werden. Klar geregelte Aufgaben und Arbeitsabläufe für jede Nutzergruppe (und damit für jeden Nutzer) sind für einen reibungslosen Betrieb unerlässlich. Eine passgenaue Zugriffsregelung für die Nutzer ist notwendig und muss gewährleistet sein.
Einzelne Einträge im Sendeplan, sogenannte Skripte, können mit Teleprompter-Texten und Videobeiträgen versehen werden. Um Redaktionsbeiträge verfassen zu können besteht die Möglichkeit in eingegangenen Agenturmeldungen zu recherchieren. Diese sind kategorisiert in Text- und Bildagenturen sowie Video- / Internetmaterial. Redaktionelle

[1] URL: `http://www.norcom.de/de/ueber-ncpower`. [02.01.2012]

Inhalte können also mit *NCPower* zentral verwaltet und publiziert werden. Innerhalb von *NCPower* können zusätzlich Nachrichten verschickt und empfangen werden. Administrative Funktionen (zum Beispiel Benutzer- und Rechteverwaltung) stehen natürlich auch zur Verfügung.
NCPower baut auf verschiedenen Ansichten auf, sodass man sich individuell seine Arbeitsumgebung erstellen kann. Eine Auflistung der Suchergebnisse (Suchliste) sowie die Detailseite eines einzelnen Ergebnisses (Vollansicht) werden zum Beispiel in separaten Fenstern dargestellt. In diesen Fenstern kann dann über (fixierbare) Reiter auf andere Ergebnisse der selben Funktion gewechselt werden. Zusätzlich hat man die Möglichkeit persönliche Hotkeys zu belegen, um die Funktionen mit denen man oft arbeitet, schnell zu erreichen. Diese Einstellungen werden für jeden Nutzer separat gespeichert.
Die Informationen zu den Funktionen von *NCPower* (Kapitel 3.1.1) stammen hauptsächlich aus einer Schulung zum Umgang mit dem Redaktionssystems. Zusätzlich wurde mit den *Mpower*-Dokumentationen gearbeitet. Die Schulung wurde im Rahmen des vorangegangenen Projekts (Kapitel 2) durchgeführt. Informationen zu dem Aufbau und der Systemarchitektur (Kapitel 3.1.2) stammen aus einer weiteren Schulung, sowie aus Gesprächen mit Mitarbeitern der CBC GmbH.

3.1.1 Funktionsumfang

Agenturmeldungen

Die Agenturmeldungen werden von dem Service „Agenturspooler“ in das Redaktionssystem übernommen[2]. *NCPower* bietet einem die Möglichkeit über den Text- und Bildagenturen zu suchen. Diese Funktion erlaubt die Verwendung verschiedener Filter, sowie der Anzeige der aktuellsten oder explizit Blitz- / Eilmeldungen. Die Filter reichen von der Stichwort- über die Freitextsuche, hin zu Einschränkungen wie eine bestimme Agentur oder einen Zeitraum. Die Ergebnisse der Suche werden in einem separaten Fenster, der Suchliste, verkürzt angezeigt. Eine Vollansicht der einzelnen Einträge wird wiederum in einem neuen Fenster dargestellt. Die Suchliste und die Vollansicht bilden die Hauptfenster von *NCPower* und werden auch für das Videomaterial in der selben Art und Weise verwendet.

Video- und Internetmaterial

Zusätzlich zu einer Suche über Text- und Bildagenturen kann auch über Video- und Internetmaterial gesucht werden. Die Filtereinstellungen für die Videorecherche sind umfassender als für die Agenturmeldungen. Die Suche kann unter anderem zusätzlich für einen bestimmten Ort, Typ oder Status des Videos eingeschränkt werden. Aus *NC-*

[2] Vgl. [Max03], S. 5

Power heraus kann direkt ein Studio-Produktionstool für das Video aufgerufen werden. So können die Redakteure das entsprechende Video direkt bearbeiten.

Sendepläne

Beim Anlegen eines Sendeplans werden Daten wie Titel, Regie, Schlagzeile, Datum, geplanter Beginn, geplante Länge, geplantes Ende, etc. angegeben. Die eigentliche Länge und damit das Ende werden aus den Zeiten für die einzelnen Skripte berechnet, sodass der Sendeplan framegenau erstellt werden kann. Zusätzlich wird farblich hervorgehoben angegeben, ob und wie viel man über / unter der Zeit ist. Es können im Sendeplan beliebig viele Zeilen erstellt werden, denen Skripte und / oder Videos zugeordnet werden. Jede Zeile des Sendeplans, also jeder Eintrag, wird vom Chef vom Dienst (CvD) freigegeben und kann dann vorerst nicht mehr geändert werden. Für die Einträge des Sendeplans gibt es verschiedene Ansichten. Neben dem gesamten Sendeplan kann man Einträge die einem selbst zugeordnet sind oder aber gelöschte Einträge (Papierkorb) anzeigen lassen[3]. Damit der Grundaufbau für bestimmte Sendungen nicht immer wieder neu erstellt werden muss, hat man außerdem die Möglichkeit Templates zu erstellen und diese immer wieder zu verwenden.
Die Berechtigungen für die Arbeit mit den Sendeplänen sind in der Regel für normale Redakteure stark eingeschränkt. Dadurch ist gewährleistet, dass nicht versehentlich Skripte von anderen Redakteuren gelöscht oder anderweitig falsche Einstellungen vorgenommen werden. Hat man eine der höheren Berechtigungsstufen, so hat man zum Beispiel Zugriff auf die Ansteuerungsmethoden der verfügbaren Teleprompter[4] oder kann den Sendeplan „On Air“ setzen.

Skript-Editor

In dem Skript-Editor werden die Texte für den Teleprompter eingegeben. Die Zeit, die voraussichtlich für den Text im Sendeplan gebraucht wird, wird automatisch berechnet. Weiterhin gibt es eine Versionsverwaltung um auf ältere Versionen des aktuellen Skripts zuzugreifen. Skripte werden entweder für eine spätere Verwendung in der persönlichen Ablage oder aber direkt in einem Sendeplan gespeichert[5].

Ablage

Den Nutzern von *NCPower* steht eine persönliche Ablage zur Verfügung. In der Ablage können entweder Skripte oder Agenturmeldungen für eine spätere Verwendung gespeichert werden.

[3] Vgl. [RTL03], S. 17
[4] Vgl. [RTL03], S. 16
[5] Vgl. [RTL03], S. 15

Nachrichtensystem

NCPower hat ein internes Nachrichten-System. Man kann anderen Nutzern Nachrichten schicken und von diesen Nachrichten empfangen. Im Produktivbetrieb wird diese Funktion allerdings kaum genutzt.

Administrative Funktionen

Natürlich bietet *NCPower* auch administrative Funktionen an. Es ist möglich Nutzer anzulegen, zu verwalten, zu löschen, usw. Dazu kommt noch die Rechteverwaltung für Nutzer- / Rollengruppen.

3.1.2 Architektur und Aufbau

NCPower ist nach einer 3-Tier Architektur aufgebaut, welche in Abbildung 3.1 dargestellt ist. Es besteht aus mehreren Clients die auf einen Server zugreifen. Dieser wiederum greift auf einen Datenbank-Server zu, auf welchem Tamino XML Server 4.4.1 läuft. Die Kommunikation und der Datenaustausch findet zwischen den einzelnen Tiers via HTTP / XML statt. Innerhalb von *NCPower* wird über den sogenannten XML-Bus kommuniziert. Für den Zugriff auf die Datenbank wird die Tamino Java API verwendet.
In *NCPower* sind die unterschiedlichen Funktionen in logische Gruppen aufgeteilt, in die sogenannten Services. Physikalisch gesehen sind die Services unabhängige Programme, die gemeinsame Merkmale haben. Dazu zählt insbesondere die XML-Bus Implementierung und damit die Anbindung an *NCPower*. Die Services werden in *NCPower* registriert sobald sie gestartet wurden und sind somit auch für den Nutzer verfügbar. Zu jedem Service gibt es eine Collection in Tamino XML Server, in welcher die zugehörigen XML-Dokumente gespeichert sind. In Tabelle 3.1 sind die Services und ihre Funktion aufgelistet. Die Services entsprechen im Großen und Ganzen den oben aufgeführten Funktionen von *NCPower*.

3.2 Anforderungen an das Redaktionssystem

Da das Redaktionssystem im Produktivbetrieb essentiell für den reibungslosen Sendeablauf ist, darf es im Betrieb zu keinerlei Fehlern oder Verzögerungen kommen. Um die Wirkzusammenhänge nocheinmal zu verdeutlichen, lohnt ein kurzer Vergleich von *NCPower* mit einem Werksfließband. Steht dieses Fließband einer beliebigen Produktion für 10 Sekunden still, so verzögern sich auch alle folgenden Fabrikationsprozesse. *NCPower* muss einen Betrieb mit 500 Redakteuren / Nutzern gleichzeitig ohne große Performanceeinbußen aushalten können.

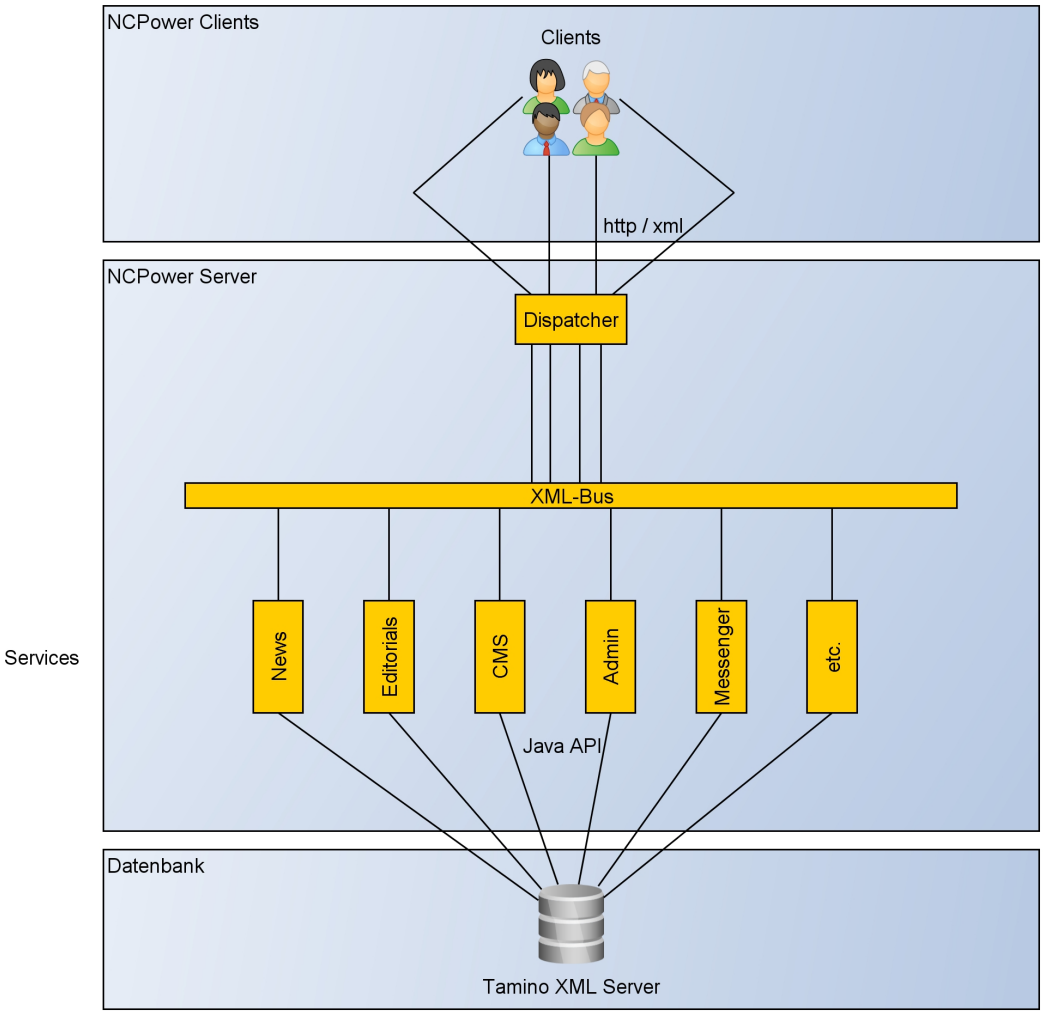


Abbildung 3.1: NCPower Architektur adaptiert nach einer Skizze von Dirk Saremba [CBC]

Es bestehen aber nicht nur generelle Anforderungen wie Performance und Ausfallsicherheit, sondern auch Anforderungen durch die verschiedenen Nutzer(-gruppen). Im Folgenden werden nur ermittelte Anforderungen aufgelistet, die sich auf die Datenhaltung auswirken. Qualitätsanforderungen, die im Entwicklungsprozess der Software berücksichtig werden müssen - wie zum Beispiel ein hohes Maß an Gebrauchstauglichkeit, eine gut strukturierte Benutzeroberfläche, ein angemessener Funktionsumfang, etc. - werden nicht aufgelistet. Die Datenhaltung wirkt sich nur indirekt respektive über andere Anforderungen auf die nicht aufgelisteten Qualitätsanforderungen aus. Einerseits beeinflusst eine schlechte Performance von Datenabfragen natürlich die Gebrauchstauglichkeit negativ, andererseits behebt eine sehr gute Performance keine schlecht struktu-

Service	**Funktion**
Admin	Einstellungs-, Nutzer-, Gruppen- und Rechteverwaltung
Archive	Archivierung von Sendeplänen
CMS	Verwaltung von multimedialen Inhalten (Videos, etc.)
Editorials (und Editorials.Versioning)	Bearbeiten von Sendeplänen inklusive Versionsverwaltung
Messaging	internes Nachrichtensystem
News	Suche über Agenturmeldungen
Pool (und Pool.Versioning)	Ablagen inklusive Versionsverwaltung

Tabelle 3.1: Übersicht der *NCPower*-Services und deren Funktionen

rierte Benutzeroberfläche. Folgende Auflistung dient als Übersicht, über die wichtigsten Anforderungen an das Redaktionssystem:

- Performance
- Robustheit
  - Ausfallsicherheit
  - Verfügbarkeit
- Zuverlässigkeit
- Daten-Backup
- Replikationen
- Wartbarkeit /Administration
- Monitoring

3.3 Anforderungen vom Redaktionssystem an die Datenbank

Die Anforderungen von *NCPower* an die Datenbank beziehen sich hauptsächlich auf den Umgang mit XML-Dokumenten / -Teilfragmenten und nur zum Teil auf Datenbankeigenschaften respektive besondere Funktionen der Datenbank. Ein Trace zwischen *NCPower*-Server und Tamino XML Server 4.4.1 hat einige dieser Anforderungen hervorgebracht. Es werden zum Beispiel für das Einfügen von Daten Transaktionen verwendet oder Cursor für das Abrufen. Die folgende Auflistung gibt eine Übersicht über die ermittelten Anforderungen:

Datenhaltung

- Transaktionen
- Cursor
- Sessions / Multiuserbetrieb
- Verfügbarkeit der Daten

Speichern

- Speichern von XML-Dokumenten
- Indexierung von XML-Dokumenten
- Geschwindigkeit (nicht so wichtig, wie bei der Suche)
- Möglichkeit mehrere XML-Dokumente gleichzeitig zu speichern (Parallelität)

Suche

- Schnelligkeit
- Suche nach bestimmten Attributen (Datum, etc.)
- Volltextsuche

Lesen / Abfrage von Daten

- XML-Dokumente müssen vollständig wiederherstellbar sein
- XML-Dokumente müssen in einen Tamino-Wrapper eingebettet sein

- Wurzel-Element des abgerufenen XML-Dokuments enthält *ino:id* als Attribut
- Geschwindigkeit der Abfrage

Aktualisieren von XML-Dokumenten

- Existierende Daten mit einer neueren Version überschreiben
- Gegebenenfalls nur Teile des Dokuments aktualisieren (produktiv nicht genutzt)

Löschen

- Löschen ganzer XML-Dokumente

4 Untersuchung der Datenbanken

Um fundierte Aussagen über die Nützlichkeit eines Mapping-Verfahrens für *NCPower* machen zu können, müssen die Datenbanken und die Organisation der Datenhaltung genauer untersucht werden. Es ist wichtig zu wissen, was die aktuelle Datenbank Tamino XML Server 4.4.1 leistet, wie die Daten abgelegt werden, und welche Möglichkeiten und Funktionen zur Verfügung stehen. Ebenso wichtig wie die derzeit eingesetzte Datenbank ist aber auch die Datenbank, die in Zukunft verwandt werden soll, Microsoft SQL Server 2008. Auch für diese Datenbank muss untersucht werden, was es für Möglichkeiten und Funktionen gibt um die XML-Dokumente bestmöglich für *NCPower* abzulegen. Zusätzlich ist eine ausführliche Untersuchung der einzelnen Collections von Tamino notwendig. An diese bestehen aufgrund verschiedener Nutzungskontexte unterschiedliche Anforderungen. Durch die Analyse der Collections können auch strukturelle Besonderheit erkannt werden.

4.1 Tamino XML Server

4.1.1 Allgemeines

Tamino XML Server 4.4.1 ist eine native XML-Datenbank von der Software AG[1]. Das heißt, die Daten werden im Dateisystem direkt als XML-Dokument abgelegt und nicht gemapped. Es handelt sich dabei also um eine dokumentorientierte Datenbank[2].
Tamino bietet einem die Möglichkeit die Daten mit zwei verschiedenen Anfragesprachen (mehr dazu in Kapitel 4.1.3) abzufragen. Weiterhin können die Daten nach Zusammengehörigkeit in Collections gespeichert und durch ein optionales Schema beschrieben werden. Dazu kann man XML Schema (XSD - XML Schema Definition) gemäß W3C[3] verwenden oder man importiert eine DTD (Document Type Definition). Das Schema kann um tamino-eigene Informationen (TSD - Tamino Schema Definition[4]) erweitert werden um die XML-Struktur noch genauer zu beschreiben und Indizes zu setzen. Im

[1] URL: `http://www.softwareag.com/de/`. [03.01.2012]
[2] Vgl. URL: `http://de.wikipedia.org/wiki/XML-Datenbank`. [03.01.2012]
[3] URL: `http://www.w3.org/XML/Schema`. [03.01.2012]
[4] mehr Informationen zu TSD in [AG11], Kapitel *Tamino XML Schema Reference Guide*

Schema wird definiert, ob die XML-Dokumente dem geschlossenen oder dem offenen Inhaltsmodell folgen sollen.

> „Bei *geschlossenem Inhaltsmodell* muss ein Dokument genau der Schemabeschreibung entsprechen, also den Validierungsregeln, wie sie in XML Schema definiert sind. Bei *offenem Inhaltsmodell* müssen die Teile des Dokumentes, die im Schema beschrieben sind, dieser Schemabeschreibung auch genügen. Es ist jedoch zulässig, dass das Dokument Elemente und Attribute enthält, die im Schema nicht beschrieben sind.“ ([Sch03], S. 274)

Tamino bietet außerdem auch noch die Möglichkeit Sessions, Cursor und Transaktionen (inklusive verschiedener Konsistenzebenen) zu erstellen und zu verwalten. Diese Funktionen sind bei nativen XML-Datenbanken im Gegensatz zu relationalen Datenbanken nicht selbstverständlich. Relationale Datenbanken sind ausgereifter, als die im Vergleich jüngeren nativen XML-Datenbanken. Das hat zur Folge, dass das Funktionalitätsspektrum im Bereich der nativen XML-Datenbanken sehr stark schwankt. So gib es aktuelle native XML-Datenbanken, die zum Beispiel nur XPath als Abfragesprachen anbieten. Andere wiederum sind sehr weit entwickelt und bietet einen umfassende - fast an die der relationalen Datenbanken heranreichende - Funktionsvielfalt.
In Tamino gibt es dank Indexierung die Möglichkeit, die Suche auf XML-Dokumenten effizienter zu gestalten. Dabei gibt es drei wesentliche Indexe: Wert-, Text- und Strukturindex. Der Wertindex ist dabei der Standardindex von Tamino, wie man ihn auch von relationalen Datenbanken kennt. Der Strukturindex dient zur Optimierung strukturbasierter Anfragen[5]. Der Textindex verbessert die Volltextsuche.

> „Dieser [der Textindex, d. Verf.] beinhaltet Worte [...] eines Textes (des Wertes von Elementen oder Attributen) und deren Kombinationen als Indexeinträge.“ ([Sch03], S. 279)

Der Zugang zu Tamino erfolgt über HTTP. Dabei werden hauptsächlich die Methoden GET und POST verwendet. Bei GET werden Anfragen und Daten als URL-Parameter und bei POST als *multipart form data* transportiert. Allerdings bietet einem Tamino auch die Möglichkeit über diverse APIs[6] auf die Datenbank zuzugreifen. *NCPower* verwendet, da es in Java geschrieben ist, die Tamino JavaAPI. Operationen werden über verschiedene Parameter gesteuert. Zum Beispiel werden mit dem Parameter *_admin* administrative Kommandos an die Datenbank geschickt, die Parametern *_cursor*, *_sessionid* und *_sessionkey* dienen zum Zugriff auf einen bestimmte Cursor, usw. Die Software AG stellt noch weitere Module für die Anpassung von Tamino bereit. So kann

[5] Vgl. [Sch03], S. 179 ff.
[6] Application Programming Interface

man Tamino über *X-Node* mit anderen Datenbanken verbinden oder Tamino Server Extensions mit *X-Tension* entwickeln, implementieren und administrieren.
Als Besonderheit ist noch das pseudo-Attribut *ino:id* zu nennen. Über diese Identifikationsnummer können bestimmte XML-Dokumente direkt angesprochen werden. Zum Beispiel gibt es zum Einfügen und Updaten von Daten, das Kommando *_process*. Wird damit ein XML-Dokument an die Datenbank geschickt, in dem keine *ino:id* vorkommt, so wird es als neues Dokument eingefügt und bekommt eine neue *ino:id* zugeordnet. Steht in dem Wurzel-Element des einzufügenden XML-Dokuments allerdings eine *ino:id* als Attribut, so wird ein vorhandenes Dokument mit der gleichen *ino:id* überschrieben. Gibt es kein Dokument mit dieser *ino:id*, so wird ein Fehler zurückgegeben[7]. Das Besondere an der *ino:id* ist aber die Tatsache, dass sie nicht persistent im XML-Dokument gespeichert ist. Bei einer Query sind Filter auf die *ino:id* als Attribut allerdings trotzdem möglich. Zusätzlich wird sie als Attribut an das Wurzel-Element der Ausgabe gehangen.

4.1.2 Organisation der Datenbank

Die Organisation einer Tamino Datenbank ist in Abbildung 4.1 dargestellt. Eine Tamino Datenbank besteht aus einer oder mehreren Collections. Dabei sollte eine Collection zusammengehörige XML-Dokumente beinhalten. Eine Collection wiederum beinhaltet einen oder mehrere Doctypes. Die XML-Dokumente sind letztlich den Doctypes zugeordnet, wobei ein Doctype XML-Dokumente mit demselben Wurzelelementtyp beinhaltet[8]. Im konkreten Fall von *NCPower* gehört eine Collection zu einem *NCPower*-Service (mehr zu den Services in Kapitel 3.1.2). Die XML-Dokumente eines Service entsprechen dem gleichen XML-Schema, können aber trotzdem unterschiedliche Strukturen - zum Beispiel durch das Kindelement *‹xs:choice›* des komplexen Typs im XML Schema - haben. Das heißt, dass die Tamino Datenbank von *NCPower* mehrere Collections hat, welchen alle XML-Dokumente der entsprechenden Services zugeordnet sind.
Der Zusammenhang zwischen Collection, Schema und Doctype ist in Abbildung 4.2 verdeutlicht. Mehrere Doctypes können zu einem Schema gehören und eine Collection kann mehrere Schemata enthalten. Für *NCPower* gibt es pro Collection ein Schema und einen Doctype zu diesem Schema[9], da die XML-Dokumente generell den gleichen Grundaufbau (siehe Kapitel 4.3.1.1 weiter unten) haben.
Beim Speichern (und auch Abrufen) von XML-Dokumenten muss eine Collection angegeben werden. Der Doctype wird anhand des Wurzelelements automatisch bestimmt.

[7] Vgl. [AG11], Kapitel *X-Machine Programming / Requests using X-Machine Commands / the _process command*
[8] Vgl. [Sch03], S. 272
[9] Vgl. [Sch03], S. 274

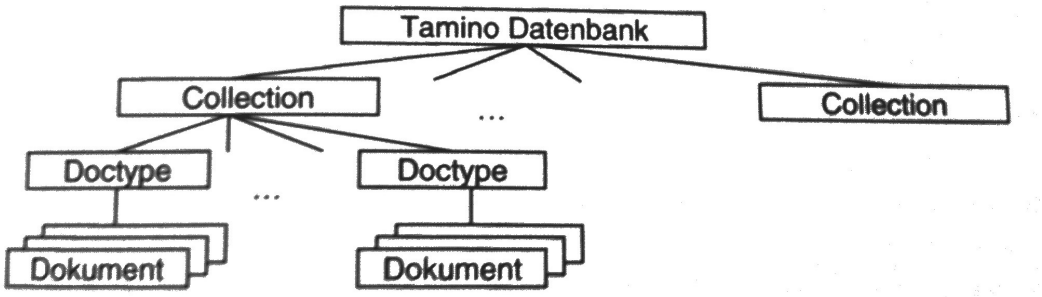


Abbildung 4.1: Organisation einer Tamino Datenbank ([Sch03], S. 273)

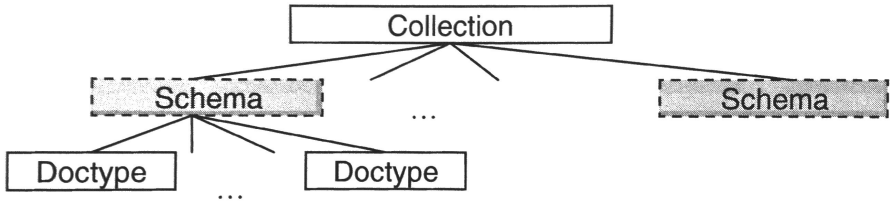


Abbildung 4.2: Zusammenhang zwischen Collection, Schema und Doctype ([Sch03], S. 274)

Beim Abspeichern wird dem XML-Dokument die *ino:id* zugeordnet. Diese ist innerhalb eines DocTypes und damit, bei der *NCPower* Tamino-Datenbank, innerhalb einer Collection eindeutig. Eine persistente Speicherung der *ino:id* innerhalb des XML-Dokuments findet nicht statt.

4.1.3 Abfragesprachen

Tamino kann über zwei Abfragesprachen angesprochen werden. Da es sich um eine native XML-Datenbank handelt, basieren die Tamino Abfragesprachen auf den XML-Abfragesprachen XQuery und XPath. Die Sprachen haben die Namen *X-Query* und *Tamino XQuery 4* und wurden ausführlich auf ihre Übersetzbarkeit in dem vorangegangen Projekt (siehe Kapitel 2) untersucht. An dieser Stelle wird auf einige wichtige Merkmale eingegangen die für die folgenden Untersuchungen von Nutzen sein könnten. Zu nennen wären dabei vor allem Funktionen und Operatoren, die hinzugefügt wurden und nicht den W3C Spezifikationen entsprechen. Da sie im Produktivbetrieb eingesetzt werden, muss darauf geachtet werden, dass die Abbildung der XML-Dokumente auf die relationale Datenbank den Einsatz dieser Funktionen nicht ver- oder behindert. Zusätzlich beeinflusst die Art der Datenhaltung natürlich auch den Zugriff auf die Daten, also das Einfügen, Aktualisieren, Löschen und Abfragen. Wird das XML-Dokument also

zum Beispiel stark fragmentiert in der relationalen Datenbank abgelegt, müsste es bei einer Abfrage per SQL gegebenenfalls über viele „Joins“ zusammengefügt werden.
Abbildung 4.3 zeigt die Tamino X-Query / XQuery 4 Spezifikationen im Kontext der W3C Spezifikationen. Anhand der Grafik kann man sehr gut erkennen, dass X-Query auf XPath und Tamino XQuery 4 auf XQuery nach dem W3C basiert. Allerdings hat die Software AG die Abfragesprachen um einige zusätzliche Funktionen erweitert und nicht alle Funktionen aus den W3C Spezifikationen übernommen.

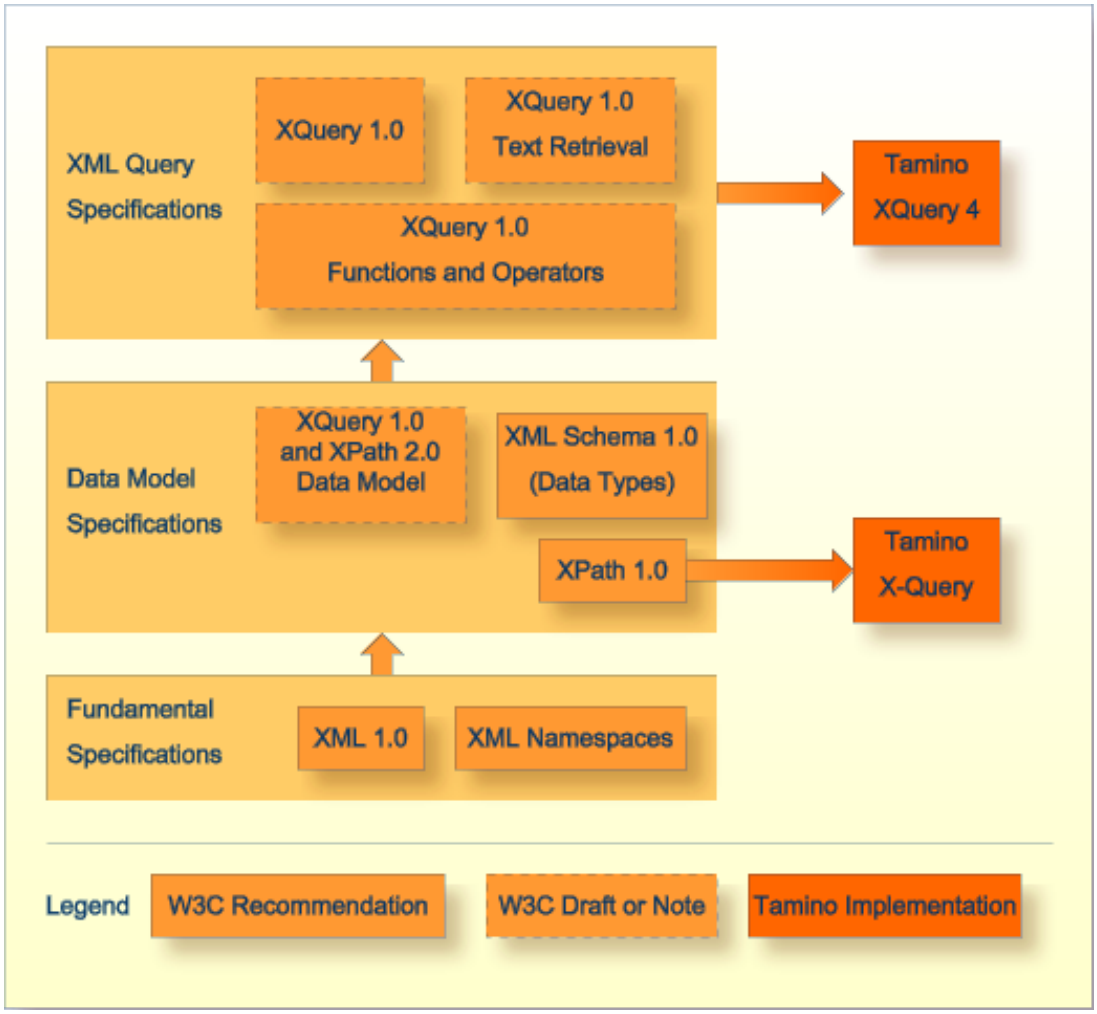


Abbildung 4.3: „Tamino XQuery 4 and W3C Specifications in Context“ aus ([AG11], Kapitel *XQuery 4 Reference Guide / Introduction*)

Auf Grund der Tatsache, dass X-Query auf XPath basiert, ist es also eine Teilmenge von XQuery. Valide X-Query Ausdrücke können also theoretisch einfach in einem XQuery Ausdruck verwendet werden. Da aber zusätzliche Funktionen und Operatoren von der Software AG implementiert wurden, ist dies nicht möglich. Als erstes sei die Sortier-

funktion *sortby()* genannt. Mit dieser lässt sich die Ergebnismenge über ein Element, Attribut oder das Ergebnis einer Funktion sortieren. XPath bietet keine Sortierfunktion, XQuery nur in Form des FLWOR-Konstrukts[10]. Die nächste wichtige Funktion ist die *contains()*-Funktion, welche in X-Query mit dem Operator „∼=“ aufgerufen wird. Es handelt sich dabei um eine Wortsuche, welche case insensitive ist und Wildcards für eine Teilwort-Suche zulässt. Im Gegensatz dazu ist das *contains()* bei XPath / XQuery eine Teilwort-Suche, die case sensitive ist und keine Wildcards erlaubt. Hier liegt auch das größte Problem, bei der Übersetzung von X-Query in SQL-Querys. Weitere zusätzliche Funktionen die implementiert wurden sind *before*, *after*, *between*, *adj* und *near*. Die Operatoren *before* und *after* dienen dazu Knoten anhand der Position der Geschwisterknoten auszuwählen. Für die Auswahl anhand eines Wertebereichs wird *between* verwendet, Wertepositionen können per *adj* (angrenzend) oder *near* (nahe) angegeben werden. Außer *after* und *before* haben die Operatoren keine Entsprechung in XPath / XQuery. Als letztes sind noch die Vereinigungsoperatoren zu nennen. X-Query bietet die Möglichkeit Vereinigungsmengen mit „|“ und Schnittmengen mit „intersect“ zu bilden. In XQuery ist dies ebenfalls möglich, zusätzlich kann auch noch „except“ verwendet werden um nur Knoten zu adressieren, die in der Menge vor dem „except“ vorkommen aber nicht in der Menge dahinter.
Für Tamino XQuery 4 sind die Unterschiede zu XPath / XQuery nicht so gravierend wie bei X-Query. Auch hier wurde die Sortierfunktion *sortby()* hinzugefügt. Diese existiert neben dem *order by*, welches bei XQuery in Form des FLWOR-Konstrukts hinzukam. Ein weiterer Zusatz in Tamino XQuery 4 ist der Operator *input()*, der vor der Pfadangabe steht. Durch ihn wird die „default Collection“ ausgewählt[11]. Die Software AG hat noch ein paar weitere Funktionen implementiert, die jedoch nicht ins Gewicht fallen und für den Sachverhalt unerheblich sind[12].
Festzuhalten ist auf jeden Fall, dass im Produktivbetrieb von *NCPower* nur X-Query im Einsatz ist. Neben Aussagen der *NCPower* betreuenden Mitarbeiter hat dies ein Trace zwischen *NCPower* und Tamino auch nochmals belegt[13]. Zusätzlich wurden so die Ausprägungen der Querys und der Aufbau der Ergebnisse deutlich.

[10] For Let Where Order by Return
[11] Vgl. [AG11], Kapitel *XQuery User Guide / Query Examples*
[12] Eine Auflistung der implementierten Tamino XQuery 4 Funktionen findet sich in [AG11] Kapitel *XQuery 4 Reference Guide / Functions Ordered by Categories*
[13] X-Query Abfragen aus dem Trace befinden sich im Anhang

4.2 Microsoft SQL Server 2008

4.2.1 Allgemeines

Microsoft SQL Server 2008 ist ein relationales Datenbankmanagementsystem (DBMS). Hierbei werden die Daten in Tabellen abgelegt. Es handelt sich also um eine datenorientierte Datenbank. Das DBMS ist mit zahlreichen Werkzeugen und Services ausgestattet, durch die es möglich ist, die Datenhaltung sehr effizient zu gestalten. Diese zusätzlichen Funktionen machen SQL Server 2008 interessanter für das Unternehmen als es Tamino ist. Bei Tamino gibt es kaum Werkzeuge, die einem helfen das System derart zu optimieren. Einige dieser Features von SQL Server 2008 sind[14]:

- Virtualisierungsunterstützung
- Replikation
- Unternehmenssicherheit
- RDBMS-Verwaltung einzelner Instanzen
- Verwaltungstools
- Analysis Services
- Berichterstellung
- etc.

Anhand der Anzahl und Charakteristik der Features von SQL Server 2008 wird deutlich, dass das System deutlich „erwachsener“ ist als Tamino XML Server. Dazu ist zu sagen, dass Microsoft SQL Server ca. 10 Jahre länger existiert / entwickelt wird als Tamino XML Server[15]. Allerdings muss man Tamino zu Gute halten, dass es dahingehend entwickelt und optimiert wurde XML-Dokumente zu speichern und zu verwalten. Bei SQL Server liegt der Fokus auf der Speicherung und Verwaltung relationaler Daten. Nichts desto trotz bietet auch Microsoft in SQL Server eine Möglichkeit zur Speicherung von XML-Daten an. Seit der Version SQL Server 2000 wird XML unterstütz und ab SQL Server 2005 gibt es sogar einen nativen XML-Datentyp[16].

[14] Vgl. [Mic10], Kapitel *Getting Started / SQL Server Installation / Overview of SQL Server Installation / Features Supported by the Editions of SQL Server 2008 R2*

[15] Microsoft SQL Server gibt es seit 1989 (URL: `http://en.wikipedia.org/wiki/Microsoft_SQL_Server`. [05.01.2012])
Tamino XML Server gibt es seit 1999 (Vgl. [Sch03], S. 271)

[16] Vgl. [Col08], S. 1 und S. 13

Zum nativen Datentyp für XML gehören einige Methoden für das Arbeiten mit XML-Strukturen. Innerhalb einer SQL Query kann zum Beispiel über bestimmte Methoden XQuery (und damit auch XPath) verwendet werden, um Querys über die XML-Strukturen durchzuführen. Eine andere Möglichkeit wäre ein Mapping der XML-Dokumente auf eine relationale Struktur. Dieses Vorgehen ist recht weit verbreitet um mithilfe von relationalen Datenbanken XML-Dokumente zu speichern. Die meisten dieser Verfahren stammen jedoch aus einer Zeit vor einer guten XML-Unterstützung durch relationale Datenbanken. Je nach Art des Zugriffs gibt es verschiedene Vor- und Nachteile, die in Kapitel 5 neben den verschiedenen Mappingverfahren genauer untersucht werden. Falles es die Struktur der Tamino XML-Dokumente zulässt kann aber auch über eine eigene Mappingalternative nachgedacht werden. Der native XML-Datentyp von SQL Server 2008 wird im folgenden Abschnitt genauer betrachtet, um im weiteren Verlauf der Untersuchung abwägen zu können, wie er sich für die Speicherung der *NCPower*-Daten eignet.

4.2.2 Der XML-Datentyp

Der native XML-Datentyp in SQL Server 2008 ist eine einfache Möglichkeit XML-Strukturen in der relationalen Datenbank zu verwalten. Noch in SQL Server 2000 war die XML Unterstützung nicht wirklich zufriedenstellend. Die Funktionsvielfalt war sehr eingeschränkt, die XML Unterstützung basierte auf den LOB-Datentypen[17] und die Implementierung war ineffizient[18]. Seitens Microsoft hat sich allerdings einiges getan. Mittlerweile gibt es neben dem nativen XML-Datentyp auch noch XML Schema Collections, XML Indizes, XQuery und XML DML Support, HTTP SOAP Endpoints und zahlreiche interne Verbesserungen[19]. Eine XML Instanz darf in SQL Server nur maximal 2 GB groß sein.
Wird ein XML-Dokument oder ein XML-Inhaltsfragment in die Datenbank eingefügt, wird dieses auf Wohlgeformtheit überprüft. Zusätzlich kann man sich entscheiden, die XML-Datentypspalte an ein XML Schema zu binden. Der SQL Server 2008 würde dann zusätzlich das XML-Dokument gegen das Schema validieren. Es ist also möglich sowohl typisiertes als auch nicht typisiertes XML zu verwenden. Die Verwendung von XML Schema bietet unter anderem Verbesserung von Abfragen durch die Typisierung.
Die Methoden, welche auf die XML-Datentypspalte angewendet werden können sind in Tabelle 4.1 dargestellt. Durch diese Methoden bieten sich komfortable Möglichkeiten, um auf die gespeicherten XML-Dokumente zuzugreifen. Sei es von einfachen XQuery-Abfragen (*query()*), über Existenz-Abfragen (*exist()*) hin zu Modifizierungen

[17] Large Object
[18] Vgl. [Col08], S. 2
[19] Vgl. [Col08], S. 13

an den XML-Dokumenten (*modify()*). Man hat also verschiedene Möglichkeiten XML-Dokumente mit XQuery abzufragen. Kombiniert mit anderen SQL Methoden kann man so eine große Anzahl verschiedener Abfragen erstellen. Die Kombination zweier Abfragesprachen erhöht unter Umständen einerseits die Komplexität der Abfragen, andererseits bieten sich einem aber auch wesentlich mehr Freiräume. Vor allem dadurch, dass man SQL als Abfragesprache nutzen kann, ergeben sich einige Vorteile.

Method	**Description**
query()	The query() method allows you to perform an XQuery on your xml instance. The result returned is untyped XML.
value()	The value() method allows you to perform an XQuery on your xml instance and returns a scalar value cast to a SQL Server data type.
exist()	The exist() method allows you to specify an XQuery on your xml instance and returns a SQL bit value of 1 if the XQuery returns a result, 0 if the XQuery returns no result, or NULL if the xml instance is NULL.
modify()	The modify() method allows you to execute XML Data Manipulation Language (XML DML) statements against an xml instance. The modify() method can only be used with a SET clause or statement
nodes()	The nodes() method allows you to shred xml instances. Shredding is the process of converting your XML data to relational form.

Tabelle 4.1: „xml Data Type Methods“ aus ([Col08], S. 71)

Der XML-Datentyp in SQL Server 2008 bietet einige Vorteile. Allen voran sind die generellen Vorteile von SQL Server, wie zum Beispiel die administrativen Funktionen und damit Backup, Recover und Replikationen. Hinzu komm die Leistungsfähigkeit für Transaktionen und XQuery / XPath Abfragen[20]. Passend dazu kann man mit Indizes und dem SQL Server Query Optimizer die Performance der Abfragen verbessern. Argumente wie die Tatsache, dass XML und relationale Daten miteinander verknüpft werden können beziehungsweise interoperieren sind irrelevant, da im Kontext *NCPower* ausschließlich XML-Dokumente verwendet werden.
Microsoft SQL Server 2008 bietet auch spezielle XML Indizes an. Es gibt drei verschiedene Arten: Den primären, sekundären (unterteilt in PATH, VALUE und PROPERTY) und den Volltext XML Index. Bevor die Indizes erläutert werden, sei kurz etwas zu der internen Verarbeitung von XQuery gesagt. Wird eine XQuery-Abfrage auf eine XML-Instanz ausgeführt, so werden die Dokumente in relationale Tabellen zerlegt.

[20] [Col08], S. 12

Über diese Tabellen wird dann die Query ausgeführt. Dieser Vorgang geschieht zur Laufzeit und kann bei vielen und großen Dokumenten zu einer schlechten Performance führen. Wird jetzt ein primärer Index auf einer XML-Datentyp Spalte definiert, so werden die XML-Dokumente vorab zerlegt. Dadurch findet die Zerlegung nicht mehr zur Laufzeit statt[21]. Der sekundären PATH Index dient der Optimierung von XQuery Pfad Ausdrücken. Ganz besonders die *exist()*-Funktion in der SQL WHERE Bedingung profitiert davon. Der sekundäre VALUE Index ist für Abfragen sinnvoll, bei denen nach einem bestimmten Wert gesucht wird, der Name und / oder die Position des Knotens, welcher diesen Wert enthält aber nicht bekannt sind. Der dritte sekundäre Index ist der PROPERTY Index. Dieser dient dazu Abfragen zu verbessern, die mithilfe der *value*-Funktion Werte abrufen[22]. Für die Volltextsuche bietet SQL Server den Volltext Index auch für XML-Instanzen an. Wird eine Textsuche in einem XML-Dokument vorgenommen, so wird nur der eigentliche Inhalt, nicht aber Markup-Elemente wie Tags oder Attribute durchsucht.
Abschließend zum XML Datentyp in SQL Server anzumerken ist noch, dass Daten, die für eine XML Instanz gespeichert werden, nicht als eins zu eins Kopie abgelegt werden. Die Daten werden konvertiert und als 16-bit Repräsentation basierend auf dem XQuery / XPath Data Model (XDM) intern gespeichert[23].

4.3 Untersuchung der einzelnen Collections

Die Untersuchung der Collections ist insofern wichtig, da jede Collection die XML-Dokumente eines anderen *NCPower*-Service beinhaltet. Die einzelnen Services bilden die unterschiedlichen Funktionen von *NCPower*, es wird also unterschiedlich mit den Daten gearbeitet. Es ist naheliegend, dass die Anforderungen an die Daten(-haltung) von Service zu Service mehr oder weniger stark variieren. Der Aufbau der Collections sowie die Aufgabe des zugehörigen Service muss betrachtet werden, um Rückschlüsse auf die Anforderungen an die einzelnen Collections ziehen zu können.

4.3.1 Aufbau der Collections

4.3.1.1 Allgemeiner Aufbau

Der Aufbau der Collections ist an den entsprechenden Service angepasst und durch ein Tamino Schema beschrieben. Dabei folgt ein Collection dem offenen, die anderen wiederum dem geschlossenen Inhaltsmodell. Allerdings ist der grundsätzliche Aufbau der

[21] Vgl. [Col08], S. 177
[22] Vgl. [Col08], S. 181-185
[23] Vgl. [Col08], S. 64 f.

Collections gleich, wie in Bild 4.4 anhand der allgemeinen Schema-Struktur zu sehen ist. Sie beginnen mit einem Kommentar, in welchem mithilfe von TSD Informationen zum Schema und Doctype angegeben sind. In diesem Kommentar werden mithilfe des Elements *‹appinfo›* Informationen an die Applikation, welche das XML-Dokument verarbeiten soll, weitergegeben[24]. Enthalten sind Name der Collection und der Name des Doctypes. Für den Doctype wird im Element *‹content›* angegeben ob es sich um das geschlossene oder offene Inhaltsmodell handelt. Zusätzlich kann über das Element *‹systemGeneratedIdentity›* festgelegt werden, ob die systemgenerierte *ino:id*, nach dem Löschen eines Dokuments mit dieser ID, erneut verwendet werden darf. Bei allen Collections ist das Wiederverwenden der *ino:id* auf *false* gesetzt.

```
<?xml version = "1.0" encoding = "UTF-8"?>
<xs:schema xmlns:MM = "www.maximedia-technologies.com/mics"
xmlns:tsd = "http://namespaces.softwareag.com/tamino/TaminoSchemaDefinition"
xmlns:xs = "http://www.w3.org/2001/XMLSchema">
  <xs:annotation>
  <xs:import namespace = "www.maximedia-technologies.com/mics"
  schemaLocation = "mm_archive.TSD"></xs:import>
  <xs:element name = "CompleteDocument">
    <xs:complexType>
      <xs:sequence>
        <xs:element name = "BaseDocument">
        <xs:element name = "DocContent">
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Abbildung 4.4: Allgemeiner Aufbau der Tamino Schemata

Darauf folgt ein *import* von einem zusätzlichen Schema. Dieses Schema beinhaltet lediglich einen Kommentar und ein zusätzliches Attribut und wird dazu verwendet die entsprechende Collection anzulegen. Das Attribut wird an jedes Element im eigentlichen Dokument (siehe „spezifischer Aufbau der einzelnen Collections“ weiter unten) angefügt und versieht diese so mit einer eigenen ID, der *EID*. Allerdings passiert das nicht bei allen Collections.

Als nächstes kommt die Definition der Struktur des XML-Dokuments. Es besteht bei jeder Collections aus dem Element *‹CompleteDocument›*. Dieses Element beinhaltet die Sequenz der Elemente *‹BaseDocument›* (auch Referenz-Bereich) gefolgt von *‹DocContent›* (auch Inhalts-Bereich). Das Element *‹BaseDocument›* beinhaltet die am meisten verwendeten Dokument Attribute um die Schnelligkeit und Effizienz der Suche (auch durch Indexierung) zu verbessern. Diese sind nochmals aufgeteilt in *‹DocQualifier›*,

[24] Vgl. URL:`http://www.w3schools.com/schema/el_appinfo.asp`. [09.01.2012]

‹AccessRights›, *‹DocUserFields›* und *‹DocSysValues›*. *‹DocQualifier›* beinhaltet drei verschiedene Referenzen zu dem Dokument, wobei nur eine dieser Referenzen wirklich einzigartig ist[25]. In *‹AccessRights›* stehen Zugriffsrechte und *‹DocUserFields›* umfasst Eigenschaften des Dokuments, die vom Client gesetzt werden können. Zu diesen Eigenschaften gehört zum Beispiel das Element *‹DocSummary›*, welches eine Zusammenfassung des eigentlichen Dokuments (Inhalt von *DocContent/Document*) beinhaltet. Bei einigen Collections (zum Beispiel News) macht eine Zusammenfassung wenig Sinn, *‹DocSummary›* beinhaltet in diesem Fall das komplette Dokument nochmals. Der Bereich *‹DocSysValues›* umfasst Systeminformationen, den Autor, die letzte Veränderung, etc.
In dem Inhalts-Bereich *‹DocContent›* befindet sich das eigentliche Dokument (im Element *‹Document›*). Bei diesem Dokument handelt es sich um den konkreten Inhalt, welcher bei Abfragen dem Nutzer angezeigt beziehungsweise von *NCPower* verarbeitet wird. Zusätzlich befinden sich dort auch noch Informationen über das Format des Dokuments, sowie ein Zeitstempel. Querys aus einem Trace zwischen *NCPower* und Tamino XML Server (siehe Anhang) zeigen, dass Filter sowohl auf Inhalte von *‹BaseDocument›* als auch *‹DocContent›* angewendet werden.
Die einzelnen Elemente inklusive kurzer Beschreibung sind in ([Max02], S. 8 ff.) zu finden. Trotz der Tatsache, dass dieses Dokument beinahe zehn Jahre alt ist, hat sich an der Struktur der XML-Dokumente wenig geändert, sodass diese Informationen weitestgehend aktuell sind. Folgende Auflistung dient zur Übersicht, über die verwendeten Begriffe:

- Referenz-Bereich - Inhalt von *‹BaseDocument›*
- Inhalts-Bereich - Inhalt von *‹DocContent›*
- Zusammenfassung - Inhalt von *BaseDocument/DocUserFields/DocSummary*
- eigentliches Dokument - Inhalt von *DocContent/Document*

4.3.1.2 Spezifischer Aufbau der einzelnen Collections

Der Aufbau der eigentlichen Dokumente jeder Collection wird in den folgenden Abschnitten genauer untersucht. Die Collections haben dieselben Namen wie die zugehörigen Services. Der allgemeine Aufbau, also der Bereich der Meta-Daten zu dem Dokument, ist dabei bei allen Collections gleich. Lediglich die Zusammenfassung variiert ebenfalls je nach Collection und entspricht entweder dem kompletten eigentlichen Dokument oder ist eine Kurzfassung von diesem.

[25] Vgl. [Max02], S. 8

Admin

Die Admin-Collection folgt dem offenen Inhaltsmodell. Das bringt den Vorteil, dass dort mit nur einem Schema mehrere unterschiedliche Inhaltsstrukturen abgedeckt werden können. In der Collection stehen unter anderem sowohl Nutzer als auch Rollen(gruppen). Rollengruppen brauchen aber zusätzliche Elemente und Attribute, welche für die Nutzer nicht notwendig sind. Durch das offene Inhaltsmodell ist der minimale Aufbau der Collection vorgegeben (siehe Kapitel 4.3.1.1) und individuelle Elemente können einzelnen XML-Dokumenten hinzugefügt werden, ohne das Schema zu verletzen. Das macht es schwierig Aussagen über die genaue Struktur der XML-Dokumente zu treffen, da diese unterschiedlichste Ausprägungen haben können. Aus diesem Grund werden Kategorien ermittelt, in die sich die verschiedenen XML-Dokumente der Admin-Collection einordnen lassen könnten.
Auffällig am Schema ist, dass das Element *‹DocContent›* ein komplexer Typ (*‹xs:complexType›*) ist, bei dem aus einer Liste von Alternativen (*‹xs:choice›*) gewählt werden kann. Es können 0 bis n (*minOccurs = "0" maxOccurs = "unbounded"*) Elemente aus den drei vorgegebenen Elementen *‹Attributes›*, *‹Document›* und *‹DocumentFormat›* vorkommen. Das Hauptelement *‹Document›* beinhaltet das eigentliche Dokument. Wobei das Schema nur einen groben Aufbau für das optionale Element *‹USER›* vorgibt. Hierbei kann ein Großteil der Übergeordneten Elemente 0 bis n mal vorkommen, es ist also so gut wie alles optional. Eine feine Granulierung findet nicht statt, es sind oftmals lediglich nur Attribute wie *name* oder *value* zu einzelnen (optionalen) Elementen vorgegeben. Durch das offene Schema und die Tatsache, dass die Hauptelemente 0 bis n auftreten können, ist eine Vielzahl von weiteren Strukturen für das Element *‹USER›* oder beliebige andere Elemente möglich.
Für das Element *‹USER›* sind Werteindizes für die Elemente *‹ROLEGROUP›* und *‹USERGROUP›* sowie die Attribute *name*, *value*, *access* und *id* definiert. Ein Textindex liegt auf dem Element *‹NAME›*.
Die Kategorien in die sich die XML-Dokumente einordnen lassen wurden anhand der Typ-Definition innerhalb der XML-Dokumente bestimmt. Einige Kategorien umfassen nur ein einziges XML-Dokument (zum Beispiel Funct, General, Export), andere wiederum beinhalten sehr viele (User, Usergroup, Keystrokestate, FavoriteWindowState, etc.). Allerdings gab es auch XML-Dokumente, die sich gar nicht kategorisieren ließen. Es werden nicht nur Nutzer und Rechte gespeichert sondern auch Einstellungen wie Fensteranordnungen oder Hotkeys für einzelne Nutzer. Anhand der folgenden Auflistung kann man das Spektrum der verschiedenen Dokumente erkennen, die alle in der Admin Collection liegen und gegen das Schema der Collection validieren.

- Actionlist
- Coolbar
- Device
- Devicemap
- Export
- Funct
- General
- Group
- FavoritesWindowState
- Keystrokestate (für jeden Nutzer)
- Navigationtree
- RDFormat
- RDView
- Rolegroup
- Rundown
- Rundowns
- Schema
- Script
- Search-Filters
- Speakers
- User
- userGroup
- VPMS
- undefiniert (rdf listen)

Archive

Die Collection Archive folgt im Gegensatz zu der Collection Admin dem geschlossenen Inhaltsmodell. Die XML-Dokumente müssen also genau dem Schema entsprechen. In diesem Fall ist das Element *‹DocContent›* ein komplexer Typ, welcher der Sequenz der Elemente *‹Attributes›*, *‹Document›*, *‹DocumentFormat›* und *‹LastModified›* entspricht. Im Gegensatz zu der Admin Collection liegt nicht nur eine Sequenz statt einer Auswahlliste vor, sondern es ist auch das Element *‹LastModified›* dazu gekommen. Zusätzlich wird über dem Element ein globaler Werteindex (Standardindex) definiert.

> „Wird ein Index auf einem globalen Element definiert, so gilt diese Definition zunächst für alle Pfade, auf denen das globale Element erreicht werden kann.“ ([Sch03], S. 280)

Das eigentliche Dokument ist entweder ein Script oder ein Sendeplan. Für das Script ist anzumerken, dass auch hier Elemente vorhanden sind, die 0 bis n-mal auftreten können. Zusätzlich kommen aber auch andere Vorgaben für die Anzahl, wie 1 bis n und 28 bis 32 vor, die aber jeweils nur einmal auftreten. Hinzu kommt, dass auch hier einige Indizes definiert wurden. Die Elemente *‹Type›*, *‹creationDate›*, *‹Status›* und *‹Title›* sowie die Attribute *group* und *uid* sind mit einem Werteindex belegt. Über den Elementen *‹Title›*, *‹Author›* und *‹CreatedBy›* liegt ein Textindex. Auch bei Sendeplänen gibt es einige Elemente die 0 bis n-mal auftreten können. Hinzu kommen einige Elemente die optional sind und jeweils ein Element das 1 bis n-mal und 0 bis 2-mal auftreten darf. Ähnlich wie bei dem Script sind auch hier einige Indizes definiert. Werteindizes liegen auf den Elementen *‹type›*, *‹uid›*, *‹creationDate›*, *‹createdBy›*, *‹lastModifiedBy›*, *‹rundownDate›*, *‹rundownTime›*, *‹startDate›*, *‹startTime›*, *‹titel›*, *‹status›*, *‹user›*, *‹userGroup›* und *‹cb›* sowie den Attributen *group* und *uid*. Zusätzlich sind über den Elementen *‹RDFormat›*, *‹title›* und *‹author›* Textindizes definiert.

CMS

Die Collection CMS folgt auch dem geschlossenen Inhaltsmodell. Genau wie bei der Collection Archive besteht auch bei CMS das Element *‹DocContent›* aus einer Sequenz der Elemente *‹Attributes›*, *‹Document›*, *‹DocumentFormat›* und *‹LastModified›* (inkl. Werteindex). Das Dokument kann auch hierbei verschiedene Ausprägungen haben. Das Element *‹Document›* besteht aus einem der folgenden Elemente: *‹VOS›*, *‹VIZMOS›*, *‹SOLO›*, *‹GRAPHICS›*, *‹SRREQ›* oder *‹ORADMOS›*. Die einzelnen Elemente zeigen keine neuen Besonderheiten.
Das Element *‹VOS›* beinhaltet mehrere optionale Elemente und jeweils ein Element das 0 bis n-mal und 0 bis 5-mal auftreten darf. Über dem Attribut *uid* wurde ein Werteindex definiert und über den Attributen *Slug*, *SlugEx* und *objShotAgencyID* liegen

Textindizes. Weitere Textindizes sind für die Elemente *‹OBJID›*, *‹OBJSLUG›*, *‹OBJGROUP›*, *‹STATUS›*, *‹DESCRIPTION›*, *‹objLoresClipStatus›* und *‹VOSSTATUS›*. *‹VISMOS›* besteht aus drei bis vier Elementen, eines davon (*‹UPLINK›*) ist optional. Ansonsten wird noch ein Werteindex für das Attribut *uid* definiert. Weitere Besonderheiten gibt es nicht. Bei *‹SOLO›*, *‹GRAPHICS›* und *‹SRREQ›* gibt es weder Einschränkungen noch Besonderheiten, wobei anzumerken ist, dass *‹GRAPHICS›* und *‹SRREQ›* identisch aufgebaut sind. *‹ORADMOS›* ist wieder ein wenig komplexer. Einige Elemente sind dort auf 0 bis 1 eingeschränkt oder 0 bis n erweitert. Werteindizes sind für die Elemente *‹Type›* und *‹itemID›* sowie für das Attribut *uid* definiert. Textindizes liegen über den Elementen *‹Tilte›*, *‹itemSlug›*, *‹objID›*, *‹mosAbstract›* und *‹objSlug›*.

Editorials inkl. Versioning

Auch die Collection Editorials folgt dem geschlossenen Inhaltsmodell. Der Aufbau von *‹DocContent›* ist dabei wie bei den Collections Archive und CMS. Anders als bei den vorherigen Collection wird allerdings für das Element *‹Document›* ein Textindex definiert. Der Inhalt von *‹Document›* ist entweder *‹Script›* oder *‹Rundown›* (Sendeplan). Das Schema enthält für *‹Script›* mehrere Elemente die 0 bis n-mal vorkommen dürfen und mehrere optionale Elemente. Werteindizes sind die Attribute *group*, *oid* und *uid* sowie die Elemente *‹TYPE›*, *‹TEMPLATE_ TYPE›*, *‹CREATIONDATE›*, *‹STATUS›* und *‹RDDATE›*. Textindizes sind die Elemente *‹TITLE›*, *‹AUTHOR›*, *‹CREATEDBY›*, *‹USER›*, *‹USERGROUP›* und *‹CB›*.

Die Schemabeschreibung zu *‹Rundown›* umfasst mehrere optionale Elemente und ein Element das 0 bis n-mal vorkommen darf (*‹RDLINE›*). *‹RDLINE›* entspricht einer Zeile im Sendeplan, also einem Skript, Video, etc. An Indizes sind zwei Textindizes auf den Elementen *‹RDFORMAT›* und *‹STARTTIME›* definiert und Werteindizes auf den Elementen *‹TYPE›*, *‹UID›*, *‹CREATIONDATE›*, *‹CREATEDBY›*, *‹LASTMODIFIEDBY›*, *‹RUNDOWNDATE›*, *‹RUNDOWNTIME›*, *‹STARTDATE›* und *‹TITLE›*.

Eine Besonderheit an der Collection ist, dass im Schema nicht nur der Doctype respektive das Element *‹CompleteDocument›* beschrieben wird, sondern auf derselben Hierarchieebene auch noch die Elemente *‹VOS›*, *‹VIZMOS›* und *‹OBJSLUG›*. Innerhalb von *DocContent / Document* beziehungsweise *BaseDocument / DocSummary* werden Elemente durch Referenzen auf diese drei Elemente definiert. Da sie öfters verwendet werden, muss so nicht bei jedem Auftreten das ganze Element definiert werden, sondern es genügt eine Referenz auf die Definition.

Die Collection Editorials.Versioning ist ganz ähnlich aufgebaut, wie Editorials. Sie dient der Versionierung und muss deshalb Daten von derselben Struktur fassen, wie Editori-

als. Allerdings muss eine Zuordnung zu anderen Versionen des Sendeplans möglich sein. Um das zu erreichen wurden an ein paar Stellen Elemente oder Attribute hinzugefügt.

Messaging

Für die Collection Messaging ist auch das geschlossene Inhaltsmodell im Schema definiert. Genau wie die vorigen Collections besteht auch hier *‹DocContent›* aus den bekannten vier Elementen. Wie bei Editorials liegt hier auf dem Element *‹Document›* ein Textindex. *‹Document›* besteht aus dem Element *‹Message›*, welches wiederrum aus 15 bis 16 weiteren Elementen besteht. Das Element *‹ATTACHMENTS›* ist dabei optional. In *‹ATTACHMENTS›* gibt es bis auf zwei optionale Elemente keine Besonderheiten. Im restlichen Dokument ist ansonsten nur ein Textindex über dem Element *‹Owner›* definiert.

News

Auch bei der News Collection gilt das geschlossene Inhaltsmodell. Der Inhalt von *‹DocContent›* ist genau, wie bei den vorherigen Collections aufgebaut. Auch hier liegt genau wie bei der Messaging Collection ein Textindex auf *‹Document›*. Das Schema gibt zwei optionale Elemente und ein Element das 0 bis n-mal vorkommen kann. Ansonsten ist ein Werteindex auf *‹STORY.DATE›* und Textindizes auf *‹MSGNUM›*, *‹TITLE›*, *‹AGENCY›*, *‹KEYWORDS›*, *‹CATEGORY›*, *‹PRIORITY›*, *‹HEADLINE›* und *‹LOCATION›* definiert.

Pool inkl. Versioning

Genau wie alle anderen Collections bis auf Admin folgt auch Pool dem geschlossenen Inhaltsmodell. Vom Aufbau her ist Pool Editorials sehr ähnlich. Es liegt auch ein Textindex auf dem Element *‹Document›*. Wie auch bei der Collection Editorials beschreibt das Schema nicht nur das Element *‹CompleteDocument›* sondern auch noch die Elemente *‹VOS›*, *‹VIZMOS›* und *‹OBJSLUG›*. Auf diese wird innerhalb der Zusammenfassung oder des eigentlichen Inhalts referenziert. Anders als bei Editorials enthält *‹DocContent›* die Elemente *‹Script›* oder *‹NITF›* (Agenturmeldung).
‹Script› enthält einige Elemente die optional sind und einige die 0 bis n-mal auftreten dürfen. Die Werteindizes sind die Attribute *group* und *uid* sowie die Elemente *‹TYPE›*, *‹TEMPLATE_ TYPE›*, *‹CREATIONDATE›* und *‹STATUS›*. Die Textindizes sind für die Elemente *‹TITLE›*, *‹AUTHOR›*, *‹CREATEDBY›*, *‹USER›*, *‹USERGROUP›* und *‹CB›* definiert.
Für das Element *‹NITF›* sind zwei optionale Elemente definiert. Die Elemente *‹MSGNUM›*, *‹PRIORITY›*, *‹STORY.DATE›*, *‹CREATIONDATE›* und *‹CREATEDBY›* sind Werteindizes und die Elemente *‹TITLE›*, *‹AGENCY›*, *‹KEYWORDS›*, *‹CATE-*

GORY›, *‹HEADLINE›*, *‹LOCATION›*, *‹USER›*, *‹USERGROUP›* und *‹CB›* sind Textindizes.
Auch bei dieser Collection sieht der Struktur der XML-Dokumente für die Versionierung Pool.Versioning sehr ähnlich aus wie die Pool Collection. Inhalte der Pool.Versioning-Collection müssen anderen Versionen derselben Inhalte zugeordnet werden können.

4.3.2 Anforderungen an die Collections

Aus den Untersuchungen ergeben sich die Anforderungen, an die Collections. Teilweise fallen diese sehr ähnlich aus, da sich die Collections selbst sehr ähneln. Hauptsächlich ergeben sich die Anforderungen aber aus der Aufgabe des zugehörigen Services. Verfügbarkeit der Daten wird nicht explizit aufgelistet, auch wenn es für einige Collections wichtig (Admin) und für andere weniger wichtig (Messages) ist. Querys aus dem Produktivbetrieb (siehe Anhang) zeigen teilweise auf, wie Inhalte angefragt werden und geben somit Hinweise auf die Nutzung.
Die Admin Collection ist essentiell für den Betrieb von *NCPower*. Da sich die Nutzer am System anmelden und ihre Rechte klar abgesteckt sein müssen, ist es besonders wichtig, dass schnell auf die Daten zugegriffen werden kann. Zusätzlich werden bei der Initialisierung von *NCPower* einige Daten aus der Collection geladen. Erschwerend kommt hinzu, dass die Ausprägungen der XML-Dokumente sehr verschieden sind. Neue Daten in der Collection entstehen beim Anlegen neuer Nutzer und Aktualisierungen treten auf, wenn Nutzer, Rechte oder Einstellungen geändert werden.
Ganz anders sieht es bei der Archives Collection aus. Hier werden Sendepläne archiviert. Lesend wird selten auf die Daten zugegriffen, gespeichert (archiviert) werden die Sendepläne regelmäßig. Solange es zu keiner Verzögerung im Produktivbetrieb durch die Archivierung gibt, müssen die Daten auch nicht hoch performant weggeschrieben werden.
Die CMS Collection beinhaltet multimediale Inhalte, Verweise und zusätzliche Informationen zu diesen. Da permanent mit multimedialen Inhalten gearbeitet wird, muss die Performance beim Lesen/Schreiben von diesen XML-Dokumenten gut sein. Zusätzlich sind in dieser Collection die meisten Datensätze vorhanden.
Da Editorials die Sendepläne beinhaltet und diese notwendig für den Produktivbetrieb sind, müssen sowohl lesende als auch schreibende Zugriffe hoch performant sein. Die Inhalte der Editorials.Versioning werden seltener gelesen. In diese Collection werden Daten gespeichert, wenn neue Inhalte erstellt oder vorhandene geändert werden.
Kaum genutzt wird dagegen der Service Messaging und damit auch die zugehörige Collection. Für Nachrichtenversand wird von Nutzern E-Mail vorgezogen. Das heißt, dass es kaum Anforderungen an die Collection Messaging gibt.

Neben Editorials ist News eine der wichtigsten Collections. Es gehen teilweise sogar im Sekundentakt mehrere Agenturmeldungen ein. Weiterhin recherchieren die Redakteure in diesen Meldungen. Es wird also mit verschiedenen Filtern über verschiedene Teile des XML-Dokuments gesucht, weshalb gerade in diesem Bereich eine optimale Leistung erzielt werden muss.
Der Service Pool ist für die Ablage zuständig. Dort „lagern“ die Redakteure ihre noch unfertigen Beiträge oder Agenturmeldungen, auf die sie noch zugreifen möchten. Die Collection ist also mit den Arbeitsabläufen verbunden. Ein flüssiges Arbeiten muss gewährleistet sein.
In einem Trace zwischen *NCPower*-Server und Tamino konnten Querys aus dem Produktivbetrieb mitgeschnitten werden (siehe Anhang). Anhand der Querys kann man sehr gut sehen, welche Teile der XML-Dokumente einer bestimmten Collection am häufigsten abgefragt werden und welche Ausprägung die Filter haben. Als Beispiel sei hier eine Query auf die Admin-Collection genannt.

```
/CompleteDocument[ BaseDocument/DocQualifier/DocReference[
 ( @uid='rolegroup/cvd_kein_TP.xml' ) or
( @uid='usergroup/edv.xml' ) ]]
```

In diesem Fall werden komplette Dokument abgerufen (*CompleteDocument*), für die eine von zwei Bedingungen gilt. Das Attribut *uid* im Element *BaseDocument/DocQualifier/DocReference* muss entweder den Wert *'rolegroup/cvd_kein_TP.xml'* oder *'usergroup/edv.xml'* haben. Betrachtet man nun mehrere Querys auf eine Collection, so erhält man Hinweise darauf, wie mit den Daten gearbeitet wird. Im Folgenden ist eine Auflistung der Anforderungen und Besonderheiten der einzelnen Collections:

Admin

- Performanter Zugriff
- offenes Inhaltsmodell
- Speicherung vieler verschieden strukturierter Inhalte
- häufiger Lesezugriff
- Veränderungen werden sporadisch vorgenommen
- Löschen von Datensätzen ist selten
- Filter größtenteils auf Referenzen in *BaseDocument*
- sehr häufiges Überprüfen des Attributs *uid* im Element *BaseDocument/DocQualifier/DocReference* auf bestimmte Werte

- selten eine Sortierung über die *ino:id* mit Ausgabe des Elements *BaseDocument*

Archive

- geschlossenes Inhaltsmodell
- regelmäßiges Speichern von neuen Datensätzen
- sehr seltene Lesezugriffe
- Aktualisierungen treten fast nie auf
- zwei verschiedene Ausprägungen
- keine Produktiv-Querys gefunden

CMS

- geschlossenes Inhaltsmodell
- häufiges Speichern von neuen Datensätzen
- häufige Lesezugriffe
- gelegentliche Aktualisierungen
- seltenes Löschen von Datensätzen
- sechs verschiedene Ausprägungen
- Abfragen sowohl von kompletten als auch von Teil-Dokumenten
- Suche/Filter sowohl in den Referenzen in *BaseDocument* als auch im eigentlichen Dokument *DocContent*
- Sortierung über *ino:id*

Editorials

- geschlossenes Inhaltsmodell
- häufiges Speichern von neuen Datensätzen
- häufige Lesezugriffe
- häufige Aktualisierungen

- Aktualisierung von einzelnen Teilen (Sendeplanzeilen) könnte sinnvoll sein
- gelegentliches Löschen von Datensätzen
- zwei verschiedene Ausprägungen
- Abfrage von kompletten und Teil-Dokumenten
- Sortierung sowohl über *ino:id* (häufig) als auch über Werte im Dokument (selten)
- Suche sowohl in den Referenzen in *BaseDocument* als auch im eigentlichen Dokument *DocContent*

Editorials.Versioning

- geschlossenes Inhaltsmodell
- häufiges Speichern von neuen Datensätzen
- seltene Lesezugriffe
- keine Aktualisierungen
- seltenes Löschen von Datensätzen
- zwei verschiedene Ausprägungen

Messaging

- geschlossenes Inhaltsmodell
- sehr seltenes Speichern von neuen Datensätzen
- seltene Lesezugriffe
- keine Aktualisierungen
- seltenes Löschen von Datensätzen
- *DocSummary* identisch mit *Document*
- Suche sowohl in den Referenzen in *BaseDocument* als auch im eigentlichen Dokument *DocContent*
- Sortierung nach *ino:id*

News

- geschlossenes Inhaltsmodell
- sehr häufiges Speichern von neuen Datensätzen
- sehr häufige Lesezugriffe
- keine Aktualisierungen
- regelmäßiges Löschen von Datensätzen
- *DocSummary* identisch mit *Document*
- verschiedene Filter über verschiedene Teile der XML-Dokumente
- sehr häufige Abfrage von Teil-Dokumenten *BaseDocument*
- Suche sehr häufig im eigentlichen Dokument *DocContent* und bei speziellen Suchkriterien auch in den Referenzen *BaseDocument*
- sehr häufige Sortierung über *ino:id*
- Wortsuche über das eigentliche Dokument *DocContent/Document*
- Verknüpfung verschiedener Suchkriterien

Pool

- geschlossenes Inhaltsmodell
- gelgentliches bis häufiges Speichern von neuen Datensätzen
- gelegentliche bis häufige Lesezugriffe
- gelegentliche bis seltene Aktualisierungen
- gelegentliches Löschen von Datensätzen
- zwei verschiedene Ausprägungen
- keine Produktiv-Querys gefunden

Pool.Versioning

- geschlossenes Inhaltsmodell
- gelegentliches Speichern von neuen Datensätzen
- gelegentliche bis seltene Lesezugriffe
- keine Aktualisierungen
- gelegentliches Löschen von Datensätzen
- zwei verschiedene Ausprägungen
- keine Produktiv-Querys gefunden

4.3.3 Ergebnisse der Untersuchungen

Die erste Erkenntnis war, dass die XML-Dokumente grundsätzlich gleich aufgebaut sind. Sie bestehen aus zwei Teilen, dem Referenz-Bereich *BaseDocument* und dem Inhalts-Bereich *DocContent*. Der eigentliche Inhalt, der von Collection zu Collection unterschiedlich aufgebaut ist, liegt in *DocContent / Document*. *BaseDocument* beinhaltet Metadaten und Referenzen um unter anderem effizienter über den XML-Dokumenten zu suchen. Zusätzlich liegt im Referenz-Bereich eine Zusammenfassung (*DocSummary*) des Inhalts. In einigen Fällen, zum Beispiel der News-Collection, unterscheiden sich Zusammenfassung und Inhalt nicht.
Es hat sich schnell herausgestellt, dass die Inhalte der XML-Dokumente der Collections unterschiedlich strukturiert sind. Die Tabellen 4.2 und 4.3 geben eine kurze Übersicht über die Untersuchungsergebnisse und zeigen die Unterschiede auf. Diese Tabellen sind allerdings nicht vollständig und dienen nur dazu, einen zusammenfassenden Eindruck von den Differenzen zu geben. Detailinformationen befinden sich in den einzelnen Abschnitten dieses Kapitels. In Tabelle 4.3 bezieht sich die Spalte *Ausprägungen* auf die unterschiedlichen Strukturen die die eigentlichen Dokumente annehmen können.
Fest zu halten ist, dass ein Schema zu einer Collection die Struktur verschiedener Inhalte beschreibt, die rein logisch zusammen gehören. Anschaulich lässt sich das am Service Editorials, also den Sendeplänen, erläutern. Ein Sendeplan besteht zum einen aus dem eigentlichen Plan welcher in der Collection Editorials als *rundown* gespeichert ist. Mit jeder Zeile des Sendeplans kann ein Skript verknüpft werden. Diese Skripte werden wiederum auch in der Collection Editorials als *script* gespeichert. Die Struktur des eigentlichen Dokuments unterscheidet sich stark, obwohl sie in derselben Collection liegen.

Collection	Inhaltsmodell	Indizes
Admin	offen	6 Werteindizes ; 1 Textindex
Archive	geschlossen	6 / 16 Werteindizes ; 3 / 3 Textindizes
CMS	geschlossen	1 / 1 / 3 Werteindizes ; 10 / 5 Textindizes
Editorials	geschlossen	8 / 9 Werteindizes ; 6 / 2 Textindizes
Messaging	geschlossen	1 Textindex
News	geschlossen	1 Werteindex ; 8 Textindizes
Pool	geschlossen	6 / 5 Werteindizes ; 6 / 9 Textindizes

Tabelle 4.2: Übersicht über die Collections

Collection	optionale / multiple Elemente	Ausprägungen
Admin	alle Elemente optional	unbegrenzt (25 gefunden)
Archive	viele 0-n ; ein 1-n ; ein 28-32 / ein 0-2	2 (Script oder Rundown)
CMS	mehrere Optionale ; mehrere 0-n ; ein 0-5	6
Editorials	mehrere Optionale ; mehrere 0-n	2 (Script oder Rundown)
Messaging	ein Optionales ; ein 15-16	1
News	zwei Optionale ; ein 0-n	1
Pool	mehrere Optionale ; mehrere 0-n	2 (Script oder NITF)

Tabelle 4.3: Übersicht über die Collections (Fortsetzung)

Auffällig ist noch, dass einige Inhaltsstrukturen in verschiedenen Collections auftreten. Alle voran sei *script* zu nennen. Skripte werden in den Collections Archive, Editorials und Pool gespeichert. Allerdings liegen diese in leicht unterschiedlichen Formen vor. Ansonsten hätte man diese gegebenenfalls auslagern und separat speichern können.

5 Strukturanaylse

Nachdem das Redaktionssystem sowie die Datenbanken untersucht wurden, besteht nun der nächste Schritt darin, verschieden Strukturalternativen und Mappingverfahren zu analysieren. Die Anforderungen an das Redaktionssystem und die Datenbank müssen durch die gewählte Struktur erfüllt werden. Im nächsten Kapitel werden mit einigen Strukturen Performance-Tests durchgeführt. Danach kann man feststellen, wie die XML-Dokumente aus Tamino am sinnvollsten in der relationalen Datenbank abgelegt werden.

5.1 Übersicht

5.1.1 Vorüberlegungen und Einschränkungen

Bevor verschiedene Strukturalternativen betrachtet werden können, sollte man sich über etwaige Vorbedingungen und / oder Einschränkungen im Klaren sein. Das erste, was geklärt werden muss, ist ob die XML-Dokumente von *NCPower* daten- oder dokumentzentriert sind. Es gibt keine wirklich klare Abgrenzung, dennoch sind einige Mappingverfahren für einen der beiden Typen prädestiniert. Die Merkmale der beiden Typen folgende[1]:

Data-Centric Documents

- XML as data transport
- Designed for machine consumption
- regular structure
- fine-grained data
- little or no mixed content
- order of sibling elements is not significant

[1] Vgl. [Bou05], S. 3 ff.

Document-Centric Documents

- Designed for human consumption
- less regular or irregular structure
- larger grained data
- lots of mixed content
- order of sibling elements is almost significant

Zumindest die ersten fünf Punkte für „Data-Centric Documents“ treffen auf die XML-Dokumente von *NCPower* zu. Man kann also festhalten, dass die XML-Dokumente, die verwaltet werden sollen, datenzentriert sind.

> „So lassen sich relationale und objekt-orientierte Datenbanksysteme besonders gut mit datenorientierten XML-Anwendungen kombinieren. Die regelmäßige und feine Struktur der XML-Dokumente ähnelt der Struktur relationaler bzw. objekt-orientierter Schemata. Die gesamte ausgereifte Technologie insbesondere der relationalen Systeme kommt zu tragen: Ausfallsicherheit, Leistungsfähigkeit, effizienter Mehrbenutzerbetrieb.“ ([KST02], S. 196)

Ein weiterer Punkt ist der Einsatz von XML Schema. Das Hauptproblem dabei ist, dass die eingesetzten XML-Strukturen in TSD (Tamino Schema Definition) beschrieben sind, SQL Server aber nur XML Schema (XSD) unterstützt. Allerdings wird im Falle *NCPower* TSD nur für zwei Erweiterungen von XML Schema verwendet. Zum einen werden dem Schema Meta-Informationen (wie Verwendung des offenen / geschlossenen Inhaltsmodells und die Wiederverwendung von der durch das System generierten ID) hinzugefügt und zum anderen werden Elemente und Attribute mit Indizes versehen. Bei der Migration zu SQL Server 2008 müssen die Indizes nicht im Schema definiert werden, sondern in SQL Server 2008 an sich. Das Inhaltsmodell hängt von der Struktur ab, die Wiederverwendung von der durch das System generierten ID ist immer deaktiviert. Sollte ein Mappingverfahren gewählt werden, welches auf dem XML Schema basiert, müssen die Schemata nur geringfügig verändert werden. Für den Einsatz in SQL Server 2008 ist keine Veränderung notwendig, da SQL Server 2008 beim Anlegen der Schema-Collection unbekannte Informationen automatisch ignoriert. Generell ist der Einsatz von XML Schema sinnvoll, da dadurch eine bessere Performance und Typisierung der Daten erreicht wird.
Problematisch wird ein Mapping für XML-Dokumente die dem offenen Inhaltsmodell folgen. Einerseits gibt das Schema nur einen „Grundaufbau“ vor, andererseits können die

XML-Dokumente unbegrenzt viele Ausprägungen haben. Nur für die Admin-Collection muss eine Lösung gefunden werden, da diese die Einzige ist, die dem offenen Inhaltsmodell folgt.

5.1.2 Strukturalternativen

Als nächstes Stellt sich die Frage, wie man XML-Strukturen in SQL Server 2008 abbilden kann. Im Folgenden wird nicht auf alle Alternativen eingegangen, sondern nur auf realistische Lösungen, die den hohen Anforderungen an das Redaktionssystem genügen könnten. Die XML-Dokumente als String in der Datenbank zu speichern und von einer Middleware parsen zu lassen um Abfragen auf sie auszuführen ist zum Beispiel nicht Zielführend.
Als erstes sei deshalb der XML-Datentyp zu nennen. Er bietet die Möglichkeit XML-Dokumente nativ abzulegen. Das heißt also, dass diese von SQL Server nicht als String gespeichert werden, der bei jedem Aufruf erst geparsed werden muss. SQL Server stellt nicht nur einige Methode bereit um auf die XML-Dokumente zuzugreifen, sondern bietet auch Optimierungsmaßnamen wie spezielle XML Indexierung an. Unter anderem kann man auch XQuery in Funktionen für den XML-Datentyp verwenden.
Eine weitere Möglichkeit ist die Abbildung der XML-Dokumente auf relationale Tabellen durch ein sogenanntes Mapping. Dabei wird das XML-Dokument nach einem bestimmten Verfahren zerlegt und in Tabellen gespeichert. Es gibt eine Vielzahl verschiedener Mappingverfahren, die jeweils andere Vor- und Nachteile haben. Zum Beispiel könnten Abfragen auf bestimmte Elemente extrem optimiert werden, andererseits könnte unter Umständen die Rekonstruktion von kompletten Dokumenten Schwierigkeiten bereiten.
Es könnte sich auch ein hybrides Verfahren anbieten, bei dem man die XML-Dokumente zerteilt und so mehrere XML-Fragmente speichert. Die Granulierung dieser Fragmentierung kann je nach Anforderungen an die Collection feiner oder grober sein. Gerade wenn bei Abfragen geprüft wird, ob Bedingungen in einem Bestimmten Bereich des Dokuments erfüllt sind, könnten sich Performance-Vorteile ergeben. Auch das Abfragen kompletter Dokumente würde voraussichtlich nicht so große Nachteile einbüßen, wie bei einigen Mappingverfahren. Allein durch die Tatsache, dass die Tamino XML-Dokumente logisch gesehen zweigeteilt sind (Referenz- und Inhalts-Bereich), würde sich an dieser Stelle eine Fragmentierung anbieten. Andererseits könnten bei Teilfragmentierung Schwierigkeiten mit der Wohlgeformtheit und der Verwendung des XML Schemas auftreten. Das Schema müsste dann ebenfalls fragmentiert werden.
Ebenfalls möglich wäre die Verwendung vom XML-Datentyp und einem Mapping. Ein Teil der Daten wird in relationalen Tabellen mit Hilfe eines Mappings gespeichert,

während der Rest der Daten als XML-Datentyp abgelegt wird. Es gibt auch Überlegungen die Daten redundant in verschiedenen Strukturen zu speichern, sodass je nach Zugriffsart die Vorteile der einzelnen Strukturen genutzt werden würden[2]. Von einer redundanten Datenhaltung wird aber aufgrund des Mehraufwands und aus Konsistenzgründen abgesehen.
Generell gibt es also drei verschiedene Möglichkeiten, XML-Strukturen sinnvoll in SQL Server 2008 abzubilden. Allerdings gibt es für das Mapping nochmals zwei verschiedene Ansätze. Man kann die XML-Dokumente schema- oder strukturzentriert mappen. Das hybride Verfahren kann n-verschiedene Ausprägungen haben, je nach Granulierungsgrad. Die einzelnen Strukturlösungen haben jeweils unterschiedliche Vor- und Nachteile und sind für bestimmte Operationen besser geeignet als für andere.

5.2 Mappingverfahren

5.2.1 Allgemein

Bevor es native XML-Datenbanken und den XML-Datentyp für relationale Datenbanken gab, musste eine Möglichkeit entwickelt werden, XML-Dokumente in Datenbanken abzulegen. Es kam die Idee auf die XML-Dokumente zu zerlegen und so in (objekt-) relationalen Tabellen zu speichern. Ansätze und Algorithmen um das zu erreichen gibt es mittlerweile viele, sie lassen sich aber in zwei Kategorien einteilen. Grundsätzlich sind Mappingverfahren entweder struktur- oder schemazentriert. Das Problem, welches bei einem XML-zu-Relational Mapping allgemein zu überbrücken gilt ist Folgendes:

> „Due to the mismatch between the tree-structure of XML documents and the flat structure of relational tables, there are many possible storage designs. For example [...] if an element [...] is guaranteed to have only a single child of a particular type [...] then the child type may optionally be *inlined*, i.e., stored in the same table as the parent.“ ([B$^+$09], S. 6)

Die schemazentrierten Verfahren verwenden das XML Schema und / oder die DTD um anhand deren Vorgaben ein relationales Schema aufzubauen. Im Gegenzug dazu setzen die strukturzentrierten Verfahren auf die Struktur von XML-Dokumenten. Es werden zum Beispiel Knoten- und / oder Kantentabellen angelegt, in welchen die Inhalte der Dokumente gespeichert werden. Dabei können je nach Struktur des XML-Dokuments teilweise sehr viele Tabellen entstehen.
Bei einem Mapping müssen verschiedene Dinge beachtet werden. Als erstes muss eine entsprechende relationale Struktur erstellt werden, in welcher die XML-Dokumente in

[2] Vgl. [B$^+$09], S. 7

einer dem Verfahren entsprechenden Form abgelegt werden können. Weiterhin müssen XML-Dokumente, sei es vorhandener Datenbestand oder neue Daten, in diese Struktur eingepflegt werden. Bei einer Abfrage der Daten, muss die Query gegebenenfalls übersetzt und das XML-Dokument respektive die originale XML-Struktur rekonstruiert werden.
Einmal abgesehen davon wie das Mapping durchgeführt wird, stellt sich noch die Frage wo das Mapping überhaupt implementiert wird. Man könnte zum Beispiel das Mapping in einer Middleware durchführen. Eine Zerteilung der XML-Dokumente kann aber auch auf Datenbankseite stattfinden. Wie das Mapping umgesetzt wird, falls die Implementierung nicht vorgeschrieben ist, entscheidet sich anhand der einzelnen Faktoren. Eine allgemeingültige Antwort kann an dieser Stelle nicht gegeben werden.

5.2.2 Übersicht über verschiedene Mappingverfahren

5.2.2.1 Strukturzentrierte Verfahren

Für strukturzentrierte Verfahren braucht man weder XML Schema noch eine DTD. Das hat den Vorteil, dass jegliche XML-Dokumente mit diesen Verfahren gemapped werden können. Allerdings muss man durch das Nichtvorhandensein des XML-Schemas auch auf einige Informationen verzichten, die für eine Zugriffsoptimierung nützlich wären. Die Daten- respektive Tabellenstruktur kann aber zusätzlich noch manuell optimiert werden.
Der erste betrachtete Ansatz wurde von Florescu und Kossmann ([FK99]) entwickelt. Der Ansatz ist über 12 Jahre alt, wird aber in aktuellen Untersuchung wie zum Beispiel [SCCSM10] oder auch in der Literatur ([KST02]) verwendet. Er basiert auf einer Graph-Darstellung des XML-Dokuments, wobei jede Kante des Graphen als Tupel in einer einzelnen Tabelle repräsentiert wird. Diese Tabelle beinhaltet die IDs des Ausgangs- und des Zielknotens, den Namen des Zielelements, den *Grad* der Geschwisterknoten, ein *Flag* welches anzeigt ob auf einen Knoten oder einen Text-Wert verwiesen wird und einen Wert, falls der Zielknoten ein Blatt ist[3]. Laut ([KST02], S. 219) „legen die Autoren auf die Bewahrung der Elementreihenfolge und die Untersuchung von Strategien zur Speicherplatz- und Zugriffszeitoptimierung“ besonderen Wert.
Ein weiterer Ansatz ist eine Erweiterung des Kanten-Verfahrens von Florescu und Kossmann. Hierbei wird das Label-Feld der Kantentabelle horizontal partitioniert. Dadurch wird für jedes Label (Tag) eine neue Tabelle erzeugt[4]. Man erhält also je nach Struktur des XML-Dokuments viele Tabellen mit wenigen Attributen.

[3] Vgl. [FK99], S. 28
[4] Vgl. [SCCSM10], Kapitel 2.3

Zwei Verfahren die vom Grundansatz her sehr ähnlich sind, sind XRel und XParent. Bei beiden werden die XML Bauminformationen in vordefinierten relationalen Schemata gespeichert. Bei XRel werden Elemente, Attribute und Text in separaten Tabellen gespeichert. Zusätzlich gibt es noch eine vierte Tabelle *Path*, welche Dokumentpfade beinhaltet. Dort werden Pfade von der Wurzel bis zu den Elementen als Abfolge von Elementen gespeichert[5]. Das Relationale Schema von XRel sieht nach ([Y+01], S. 10) folgendermaßen aus:

```
Element (docID, pathID, start, end, index, reindex)
Attribute (docID, pathID, start, end, value)
Text (docID, pathID, start, end, value)
Path (pathID, pathexp)
```

Das relationale Schema von XParent sieht ein wenig anders aus, besteht aber ebenfalls aus vier Tabellen. *Element* enthält alle Elemente des Dokuments, während *Data* Attribute und Text-Werte beinhaltet. In der Tabelle *LabelPath* sind alle Pfade und deren Länge gespeichert, *DataPath* hingegen enthält die Eltern-Kind-Beziehungen[6]. Die einzelnen Bestandteile sind, genau wie bei XRel, durch IDs miteinander verknüpft. Im Detail sieht das relationale Schema für XParent nach ([J+01], S. 4) wie folgt aus:

```
LabelPath (ID, Len, Path)
DataPath (Pid, Cid)
Element (PathID, Did, Ordinal)
Data (PathID, Did, Ordinal, Value)
```

5.2.2.2 Schemazentrierte Verfahren

Einige der schemazentrierten Verfahren benötigen eine DTD. Sollte keine DTD zu einem Dokument vorliegen, so gibt es Verfahren um aus XML-Dokumenten DTDs zu erstellen, die an dieser Stelle aber nicht diskutiert werden. Eines der bekanntesten schemazentrierten Mapping-Verfahren ist von Shanmugasundaram et. al. ([S+99]). Dabei wird zu jedem Elementtyp eine Relation erzeugt. Das geschieht durch die Erstellung von einem DTD-Graph und daraufhin Elementgraphen.

> „A DTD graph represents the structure of a DTD. Its nodes are elements, attributes and operators in the DTD. Each element appears exactly once in the graph, while attributes and operators appear as many times as they appear in the DTD. [...] Cycles in the DTD graph indicate the presence of recursion.“ ([S+99], S. 5)

[5] Vgl. [Y+01], S. 10
[6] Vgl. [J+01], S. 4

Aus diesem DTD-Graphen wird mithilfe der Tiefensuche ein Elementgraph erstellt. Dieser dient dazu *Rücksprungkanten* für Rekursionen zu erstellen. Die Relationen werden aus diesem Elementgraphen erzeugt.

> „A relation is created for the root element of the graph. All the element's descendents are inlined into that relation with the following two exceptions: (a) children directly below a "*" [oder "+", d. Verf.] node are made into separate relations - this corresponds to creating a new relation for a set-valued child; and (b) each node having a backpointer edge pointing to it is made into a separate relation - this corresponds to creating a new relation to handle recursion.“ ([S+99], S. 5)

Eine weitere Mapping-Möglichkeit ist die Middleware RelaXML ([K+05]), die sich um das Mapping von XML-Strukturen auf relationale Tabellen (import) und die Überführung von relationalen Daten in XML-Dokumente (export) kümmert. Dafür benötigt RelaXML drei Dinge: Die Daten in relationalen Tabellen (beliebige Struktur), ein *concept* und eine *structure definition*. Letzteres ist im Prinzip das XML Schema und das *concept* entspricht einer Sicht (View) im relationalen Sinne. Durch das *concept* können SQL-Querys generiert werden um die entsprechenden Daten abzufragen. Durch Nutzer spezifizierte Transformationen kann das ResultSet falls nötig angepasst werden. Mithilfe des XML Schemas, also der *structure definition*, wird letztendlich das XML-Dokument erzeugt. Das Einfügen von XML-Dokumenten in die Datenbank erfolgt auf umgekehrtem Weg[7].
Abbildung 5.1 zeigt die Architektur des Mapping-Mechanismus des schemazentrierten Mapping-Frameworks LegoDB ([B+02]). Das Mapping wird aus drei Komponenten erzeugt, die für ein optimales Ergebnis sorgen sollen. Aus XML Schema und Statistiken, die aus einem XML Beispieldatenset erzeugt wurden, wird ein physikalisches Schema erzeugt. Diese physikalische Schema ist im Prinzip eine Erweiterung des XML Schemas um eben diese XML Statistiken[8]. Im nächsten Schritt wird mit dem erzeugten physikalischen XML Schema und einem XQuery-Workload ein relationales Schema und Statistiken, sowie SQL-Querys erzeugt. Mit einem relationalen Query-Optimierer können die Kosten bestimmt werden. Sind diese nicht zufriedenstellend, so kann das physikalische Schema durch Transformationen angepasst werden. Der Vorgang wird solange wiederholt, bis man ein zufriedenstellendes Ergebnis erhält[9].

[7] Vgl. [K+05], S. 1 f.
[8] Vgl. [B+02], S. 2
[9] Vgl. [B+02], S. 7 f.

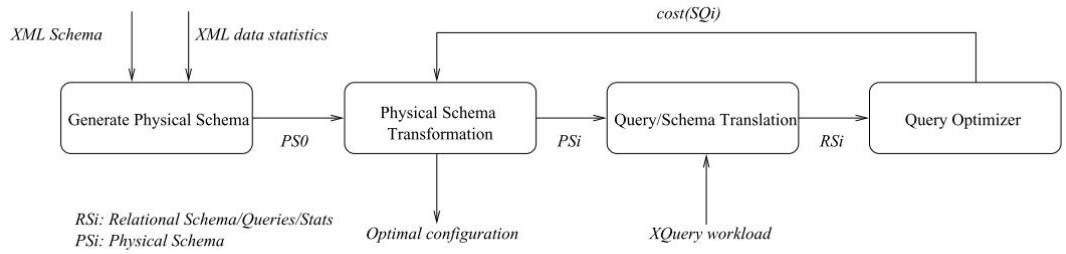


Abbildung 5.1: „Architecture of the Mapping Engine“ ([B+02], S. 8)

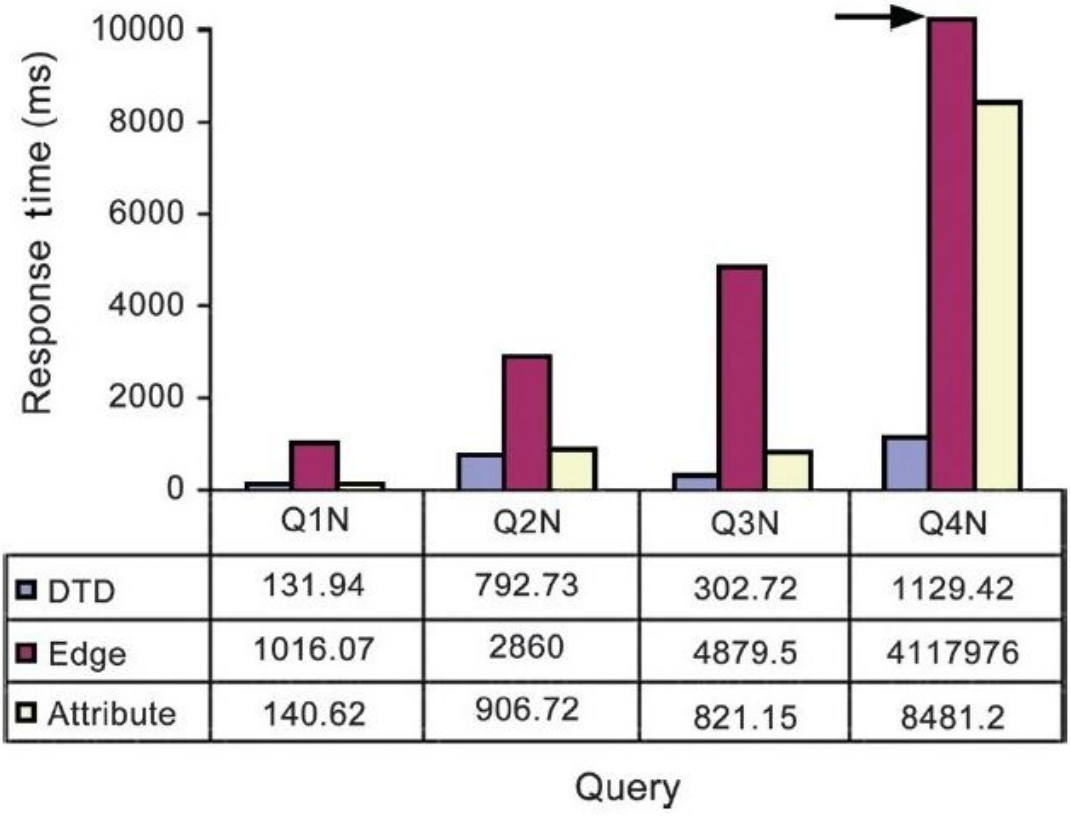
Um ein Mapping nach individuellen Vorgaben zu erstellen kann man Shrex[10] verwenden. Shrex nutzt das *‹xs:annotation›*-Element um im XML Schema Informationen zum Shredding anzugeben. Dadurch ermöglicht es eine Vielzahl verschiedener Mapping-Verfahren.

5.2.2.3 Vergleich der Verfahren

Vor allem bei dem erweiterten Verfahren von Florescu und Kossmann mit horizontaler Partitionierung des Label-Felds können sehr viele relationale Tabellen mit wenigen Spalten entstehen. Bei dem Kanten-Tabellen-Ansatz (also dem Verfahren von Florescu und Kossmann ohne Erweiterungen) dagegen entsteht eine sehr große Tabelle mit fünf Spalten. Die Verfahren XRel und XParent basieren auf einem vorgegebenen relationalen Schema (jeweils vier Tabellen). Unter anderem werden Pfade zu allen Elementen gespeichert. Bei XML-Dokumenten mit vielen Elementen und einer Verschachtelung dieser könnten die *Path*-Tabellen sehr viele Datensätze enthalten.
Bei dem schemazentrierten Verfahren von Shanmugasundaram et. al. entstehen sehr viele Tabellen. Das liegt daran, dass für jeden Elementtyp, für Wiederholbare Elemente und für Rücksprungkanten eigene Relationen erzeugt werden. Zusätzlich hat die Tabelle jedes Elementtypen (welcher selbst ein Wurzelelement sein kann) für jedes Unterelement ein Attribut. Bei stark verschachtelten XML-Dokumenten entstehen so sehr viele Tabellen mit vielen Attributen. Die Middleware RelaXML erlaubt es, die Daten in beliebiger relationaler Struktur abzulegen. Allerdings werden dadurch für das Einfügen und Auslesen von Daten zusätzliche Informationen benötigt. Dabei handelt es sich um ein Sicht-ähnliches Konstrukt und dem XML Schema. LegoDB bietet einen kostenbasierten Ansatz, bei dem mithilfe von XML Schema, Statistiken aus XML Testdatensätzen und einem XQuery Workload ein möglichst gutes relationales Schema durch Iterationen erzeugt wird.

[10] Amer-Yahia, S., Du, F., & Freire, J. (2004). A comprehensive solution to the XML-to-relational mapping problem. *WIDM 04*. Retrieved from `http://dl.acm.org/citation.cfm?id=1031461`. [24.01.2012]

In [SCCSM10] werden die beiden strukturzentrierten Ansätze von Florescu und Kossmann und das schemazentrierte Verfahren von Shanmugasundaram et al. verglichen. Dabei stellt sich bereits bei dem ersten Test heraus, dass der Kanten-Tabellen-Ansatz sehr unperformant ist. Ein Vergleich der Query Laufzeiten, von den drei Verfahren auf einem Testdatensatz, ist in Abbildung 5.2 zu sehen. Der Testdatensatz ist der Kleinste der für die Tests verwendet wurde. Er ist 23.8 MB groß und die maximale Baum-Tiefe ist 8[11]. Bestenfalls braucht der Kanten-Tabellen-Ansatz dreimal länger als der DTD-Ansatz. Nach diesem ersten Test wurde der Kanten-Tabellen-Ansatz verworfen, stattdessen hat man zusätzlich die anderen beiden Ansätze mit und ohne Indexierung verglichen.



	Q1N	Q2N	Q3N	Q4N
DTD	131.94	792.73	302.72	1129.42
Edge	1016.07	2860	4879.5	4117976
Attribute	140.62	906.72	821.15	8481.2

Abbildung 5.2: „Comparison of the query response time on the data type definition, edge, and attribute approaches based on the NASA dataset“ ([SCCSM10], Kapitel 5.2.1 - Bild 3)

In der Grafik sieht man auch, dass der Ansatz mit der horizontalen Partitionierung des Label-Felds schlechter ist, als der DTD-Ansatz. Weitere Tests mit andersartigen Datensätzen zeigten, dass der DTD-Ansatz immer bessere Ergebnisse lieferte und dass

[11] Vgl. [SCCSM10], Kapitel 4 - Tabelle 4

eine Indexierung die Laufzeit ungefähr halbierte[12]. An dieser Stelle ist allerdings anzumerken, dass der DTD-Ansatz für komplexe XML-Dokumente - wie die von *NCPower* - absolut ungeeignet ist.

> „Die Menge der generierten Relationen war für 11 der 37 [Testmenge, d. Verf.] DTDs so groß, dass das verwendete Datenbanksystem mit einem Überlauf des virtuellen Speichers abstürzte. So wurden für eine 19-elementige DTD 3462 Tabellen generiert.“ ([KST02], S. 219)

Die beiden Verfahren XRel und XParent sind von der Grundidee sehr ähnlich. Bei beiden werden die Daten auf vier relationale Tabellen verteilt. Zusätzlich zu den Elementen und deren Inhalt werden noch Pfad-Informationen gespeichert. In [J+01] werden XParent, XRel und der Ansatz von Florescu und Kossmann miteinander verglichen. Implementierungen der drei Strukturen wurden mit den gleichen Daten und Querys getestet. Abbildung 5.3 zeigt die Ergebnisse der Untersuchungen. Man sieht sehr deutlich, dass das Verfahren von Florescu und Kossmann, außer bei einer Query, wesentlich schlechter als XRel und XParent ist. Weiterhin sind XRel und XParent bei den meisten Query ungefähr gleich schnell, lediglich bei einer Query ist XParent deutlich schneller.

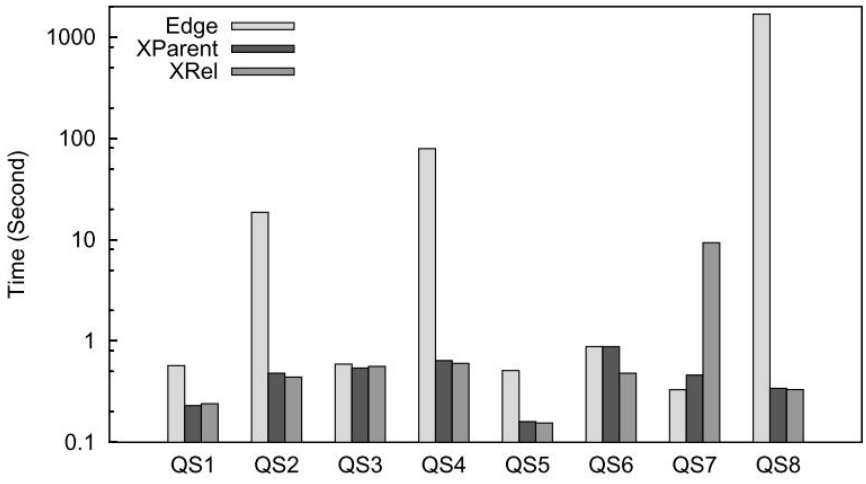


Abbildung 5.3: „Query Elapsed Time: Edge, XParent and XRel Using SHAKES“ ([J+01], S. 8)

Zusätzlich wird in dem Paper auch noch XRel mit XParent mit einem anderen Datenbestand und anderen Querys miteinander verglichen. Die Ergebnisse der Laufzeiten

[12] Vgl. [SCCSM10], Kapitel 6

sind in Abbildung 5.4 zu sehen. Bei einer Query hat XRel die besseren Laufzeiten, ansonsten ist XParent entweder besser oder beide Strukturen liefern annähernd das gleiche Ergebnis. Bei einem weiteren Test[13] wurden drei verschieden große Ausprägungen eines Datensatzes und acht Querys getestet. Im Durchschnitt ist das Verhalten von XParent besser als das von XRel. Als Besonderheit anzumerken ist, dass sowohl für XRel als auch XParent bei zwei Querys die Ausführungszeit ab einer bestimmten Größe sank und nicht stieg.

> „This is caused by a shift to relatively "better" physical query plans by the query optimizer.“ ([J+01], S. 8)

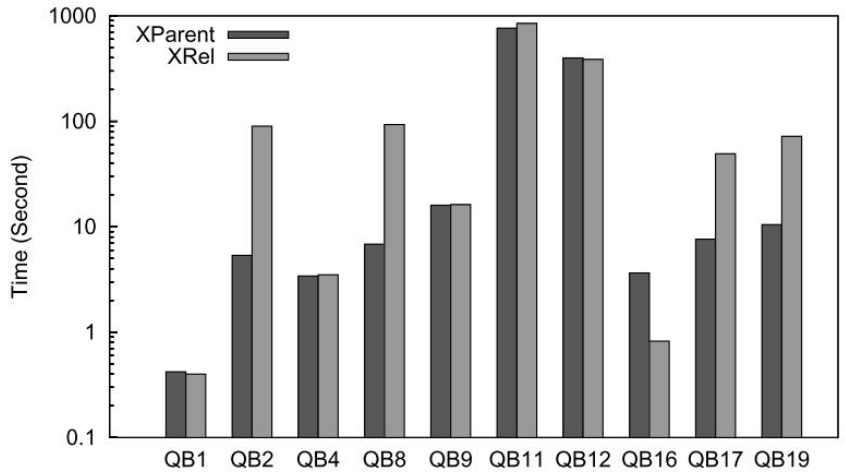


Abbildung 5.4: „Query Elapsed Time: XRel vs XParent Using BENCH0.4“ ([J+01], S. 8)

Die anderen Verfahren, RelaXML und LegoDB, lassen sich ad hoc nur schwer miteinander vergleichen. Allerdings bietet LegoDB den Vorteil der iterativen Verbesserung des relationalen Schemas und der Nutzung des Query-Optimierers. Zusätzlich übernimmt das Framework auch die Übersetzung von XQuery in SQL. Wohingegen RelaXML durch *Concept*, *structure definition* und Transformationen sehr gut an die Anforderungen angepasst werden kann. In der Literatur gibt es keine direkten Vergleiche zwischen RelaXML und LegoDB. Allerdings gibt es in den einzelnen Papern jeweils Performance-Untersuchungen.
Für LegoDB wurden nur verschiedene XML Transformationen, die Effizienz von einem *gierigen* Algorithmus zum Finden der besten Konfiguration, sowie das Verhalten, bei

[13] Vgl. [J+01], S. 9

sich im Laufe der Zeit ändernden Query-Workloads, untersucht[14]. Die Erkenntnisse aus den Untersuchungen sind für die Wahl einer Struktur für die XML-Dokumente von *NCPower* irrelevant. Bei RelaXML wird der Import und Export von Daten im Vergleich zwischen RelaXML und einer hart-codierten JDBC-Applikation verglichen. Der Vergleich ergibt, dass die Ausführungszeit bei RelaXML immer höher ist (bei 50.000 Datensätzen um den Faktor 3)[15] als bei der JDBC-Applikation.

> „Performance studies showed a reasonable overhead when exporting and importing compared to the equivalent hand-coded programs. This overhead is easily offset by the offered flexibility, simplicity, and labor-savings of RELAXML compared to hand-coded programs, e.g., in web-service applications.“ ([K+05], S. 23)

Da RelaXML im Vergleich zu einem hart-codierten Programm Overhead erzeugt, ist diese Methode nicht die optimale Lösung für das Redaktionssystem. Die gebotene Flexibilität und Einfachheit sind in diesem Fall keine Argumente um Performance-Nachteile in Kauf zu nehmen. Für andere Applikationen mag dies vielleicht der Weg der Wahl sein, für die *NCPower*-Collections allerdings nicht. In wie fern LegoDB mit den anderen Strukturlösungen konkurrieren kann muss noch herausgefunden werden.

5.3 Strukturen

Als nächstes können die einzelnen Strukturlösungen genauer auf ihre Anforderungen, Einsatzgebiete, Vor- und Nachteile untersucht werden. Durch Vergleiche der einzelnen Verfahren, kann man ein gutes Bild erhalten, welche Strukturlösungen von vorne herein ungeeignet sind oder sich potenziell eignen würden. Zusätzlich darf nicht aus den Augen verloren werden, welche Probleme auftreten könnten und ob es sich lohnen würde, diese anzugehen.

5.3.1 Anforderungen und Eigenschaften

Abgesehen von dem nativen XML-Datentyp haben die Strukturen auch Anforderungen an die Datenbank und die Art und Weise wie sie implementiert werden. Zusätzlich gibt es auch allgemeine Anforderungen an die Struktur. Diese gehen zwar mehr oder weniger aus den Ergebnissen aus Kapitel 3 hervor, seien hier aber noch einmal generell erwähnt.

[14] Siehe [B+02], Kapitel 5
[15] Siehe [K+05], S. 22 - Figure 10

Allgemein ist für das Arbeiten mit XML-Dokumenten einiges zu beachten und auch speziell in Bezug auf *NCPower* müssen einige Anforderungen an die XML-Dokumente erfüllt sein. Dabei ist es wichtig, dass diese Anforderungen auch von den Strukturlösungen unterstützt werden. Dazu gehören *query performance*, *data manipulation performance*, *space and communication efficiency*, *schema flexibility* und *miscellaneous features*[16]. Ganz besonders wichtig ist die *query performance* und - zu *data manipulation performance* gehörend - die (Teil-)Dokument-Abfragen. Dazu kommt die Optimierbarkeit der Datenhaltung zum Beispiel durch XML-Indizes. Unwichtig hingegen ist die *schema flexibility*, da sich die Schemata nicht ändern und die Struktur der XML-Dokumente gleich bleibt.
Die Mapping-Verfahren haben generell den Vorteil, dass die Daten in relationalen Tabellen liegen. Das heißt, sie können mit herkömmlichen relationalen Mitteln optimiert werden. Dem XML-Datentyp stehen neben XML-Schema noch drei verschiedene Indizes als Optimierungsmaßnahmen zur Verfügung. Offen ist, ob die relationalen Optimierungsmaßnahmen die Nachteile, die die Mapping-Verfahren mit sich bringen im Vergleich zum XML-Datentyp aufwiegen.
Die Eigenschaften der Strukturen unterscheiden sich für die unterschiedlichen Aufgaben. Erste Einschätzungen, welche Struktur sich in welchem Fall besser verhält, können an dieser Stelle nur geschätzt werden und müssten durch Tests validiert werden. Allerdings lassen sich in der Literatur Vergleiche zwischen einige Verfahren finden. In Tabelle 5.1 ist eine Vergleich zwischen verschiedenen Strukturlösungen und Eigenschaften aufgeführt. Dort wird die Speicherung von XML-Dokumenten als LOB, objekt-relational und nativ miteinander verglichen. Anzumerken ist allerdings, dass in dem Dokument eine eigene Implementierung der nativen XML-Speicherung untersucht wird und nicht der XML-Datentyp einer relationalen Datenbank. Dennoch zeigt die Tabelle einen Trend, wie gut die Speicherlösungen in den einzelnen Gebieten sind.

	LOB Storage	**OR Storage**	**Native Storage**
Query	Poor	Excellent	Good/Excellent
DML	Poor/Good	Good/Excellent	Excellent
Document Retrieval	Excellent	Good/Excellent	Excellent
Schema Flexibility	Poor	Good	Excellent
Document fidelity	Excellent	Poor	Good/Excellent
Mid-tier integration	Poor	Poor	Excellent

Tabelle 5.1: „Comparisons between LOB, OR, and Native Storages“ aus ([ZO10], S. 3)

[16] Vgl. [ZO10], S. 3

An der Tabelle sieht man ganz klar, dass LOB, bis auf bei dem Abrufen ganzer XML-Dokumente (*Document Retrieval*) und der Originaltreue dieser, am schlechtesten ist. Die objekt-relationale Speicherung ist bei der Query-Ausführung sehr gut, ansonsten erzielt aber die native XML Speicherung bessere Ergebnisse. Inwiefern jedoch der native XML-Datentyp von SQL Server 2008 mit der Untersuchung von [ZO10] vergleichbar ist, ist nicht klar.
Aber auch in anderen Untersuchungen wurden verschiedene Verfahren miteinander verglichen. In [AEN08] werden ein Struktur-Mapping, der XML-Datentyp in SQL Server 2005 mit XML Schema und fünf verschiedene hybride Verfahren anhand verschiedener Querys gegeneinander gestellt. Für den Vergleich wurden partiell-strukturierte XML-Dokumente verwendet. Die Ergebnisse können nicht a priori auf die Daten von *NC-Power* bezogen werden, da die Querys einer besonderen Semantik entsprechen müssen. Querys, die der Tamino-Semantik entsprechen, wurden in den Tests allerdings nicht verwendet. Die hybriden Verfahren dienen dazu, die Auswirkungen des Verhältnisses zwischen semi-strukturierten und strukturierten Daten eines XML-Dokuments auf die Query-Laufzeit zu untersuchen. Dabei wurde zwischen der vertikalen und der horizontalen Dimension unterschieden. Die vertikale Dimension bezieht sich auf das Verhältnis von semi-strukturierten und strukturierten Komponenten des Schemas. Wohingegen die horizontale Dimension das Verhältnis von semi-strukturierten zu strukturierten Dateninstanzen meint. Die vertikale Dimension wird auf drei verschiedene Arten variiert, nämlich mit einem Struktur-Mapping für den Dokument-Schlüssel, Dokument-Schlüssel und Autoren und Dokument-Schlüssel und Titel. Bei der horizontalen Dimension wird das Verhältnis zwischen Struktur-Mapping und XML-Datentyp in den Ausprägungen 40% / 60% und 63% / 37% verwendet[17]. Die verschiedenen Strukturen wurden mit 20 verschiedenen Querys getestet. Dabei hat sich herausgestellt, dass sowohl das Struktur-Mapping, als auch der native XML-Datentyp gegenüber den hybriden Verfahren im Großteil der Fälle im Nachteil sind. Das Struktur-Mapping wies für vier der Querys ein annehmbares Ergebnis auf, der native XML-Datentyp sogar nur für zwei[18]. Die Query-Laufzeiten wurden auch bei einer Vergrößerung der Datensätze um den Faktor 2 und 3 untersucht. Dabei fällt auf, dass bei den hybriden Strukturen mit Veränderung der vertikalen Dimension die Verschlechterung am größten ist. Allerdings lieferten diese Verfahren bei „normaler “ Datensatzgröße auch die besten Ergebnisse[19].

[17] Vgl. [AEN08], S. 2 f.
[18] Siehe [AEN08], S. 4
[19] Vgl. [AEN08], S. 3 ff.

5.3.2 Mögliche Probleme

Vor allem für die strukturorientierten Mappingverfahren ist keine Implementierung vorgegeben. Es wird lediglich das relationale Schema erläutert, nicht jedoch, wie genau XML-Dokumente in die Tabellen gebracht und wieder rekonstruiert werden. Ein Verfahren hierfür müsste erst noch entwickelt, getestet und implementiert werden. Dabei ist auf jeden Fall auf eine gute Performance zu achten, da eine noch so gute Datenhaltung keine Vorteile bringt, wenn Verarbeitung in einer Middleware unperformant ist. Aber nicht nur das Einfügen und Abrufen kompletter XML-Dokumente bedarf einem Verfahren, sondern auch die Query-Übersetzung. Diese sieht je nach gewählter Struktur anders aus.
Wird eine hybride Struktur eingesetzt, so werden die XML-Dokumente an logisch sinnvollen Stellen fragmentiert. Die einzelnen Fragmente können aber nicht ohne weiteres gegen ein XML Schema validiert werden, da das Schema nur für komplette Dokumente definiert ist. Man müsste das XML Schema also manuell an denselben Stellen fragmentierten. Der Aufwand ist aber nicht sehr groß, da es insgesamt nur neun Collections gibt und sich die hybride Struktur gegebenenfalls nicht für jede Collection eignet.
Ein weiteres Problem ist, dass unter Umständen sehr viele Daten verarbeitet und gespeichert werden müssen. Gerade wenn ein einzelnes XML-Dokument aus verschiedenen relationalen Tabellen rekonstruiert werden muss, könnte die Performance sehr stark leiden. Die Collections haben immerhin Größen zwischen 50 MB und über 5 GB.

5.4 Auswirkungen

Egal welches der Verfahren verwendet wird, man muss in jedem Fall die XPath/XQuery-Anfrage in irgendeiner Form übersetzen. Verwendet man den nativen XML Datentyp von SQL Server 2008, so müssen die Querys theoretisch nur in einem SQL Statement eingebettet werden. Allerdings hängt dies auch von der Semantik der Anfrage ab. Je nach gewünschtem Ergebnis müssen gegebenenfalls mehrere der SQL Funktionen für den XML-Datentyp verwendet werden.
Bei den Mapping-Verfahren sieht dies allerdings ganz anders aus. Je nach gewähltem Verfahren entstehen unterschiedlich viele relationale Tabellen. XPath/XQuery kann für die Abfragen gar nicht mehr verwendet werden und muss vollständig in eine SQL-Query übersetzt werden. Damit verbunden sind in einigen Fällen viele *joins*, welche zu hohen Kosten führen. Die hybriden Verfahren würden beide Verfahren miteinander verbinden. So kann auf Teile des XML-Dokuments XPath/XQuery angewendet werden, aber zur Abfrage des kompletten Dokuments müsste trotzdem gejoined werden.
Aber nicht nur das Abfragen von Daten, sondern auch beim Einfügen verändert sich je

nach Verfahren mehr oder weniger viel. Wird nur der native XML- Datentyp verwendet, so können die XML-Dokumente einfach eingefügt werden. Sollte aber das hybride Verfahren oder ein Mapping verwendet werden, so müssen die Daten entsprechend fragmentiert werden. Gerade bei den Mapping-Verfahren wird das XML-Dokument komplett zerlegt. Bei den hybriden Verfahren kann zum Teil die XML-Struktur beibehalten werden, was die Rekonstruierbarkeit erleichtert.
Vor allem Mapping-Verfahren bieten beim Updaten von Teilen oder noch besser einzelnen Werten eines XML-Dokuments Vorteile. Es muss so nicht unnötig das gesamte Dokument aktualisiert beziehungsweise durchsucht werden. Ein Sonderfall des Update ist das Ersetzen eines kompletten Dokuments und genauso werden Daten von *NCPower* in Tamino aktualisiert. Hierbei gilt das gleiche, wie für das Einfügen von Dokumenten. Das XML-Dokument müsste erst fragmentiert werden, zusätzlich müssten die entsprechenden Datensätze in jeder Tabelle überschrieben werden.
Durch ein Mapping entsteht unter Umständen beim Löschen im Vergleich zum nativen XML-Datentyp ein Mehraufwand. Da das XML-Dokument gegebenenfalls auf mehrere Tabellen verteilt ist, müssen aus allen Tabellen Daten gelöscht werden. Beim nativen XML-Datentyp wird beim Löschen einfach der entsprechende Datensatz gelöscht. Auch in diesem Fall könnte das hybride Verfahren einen Mittelweg darstellen. Es müssten zwar auch Daten aus verschiedenen Tabellen gelöscht werden, aber dies wäre je nach Granulierung respektive Umsetzung mit mehr oder weniger Aufwand verbunden.
Je nachdem wie die Daten letztendlich abgelegt werden, so ändert sich auch der Speicherverbrauch. Inwiefern der Speicherplatzverbrauch unterschiedlicher Strukturen unterscheidet wird in Kapitel 6 unter anderem betrachtet.
Die Wahl der Struktur hat definitiv Auswirkungen auf der Performance der Datenbehandlung. Der native XML-Datentyp hat den Vorteil, dass die XML-Dokumente nicht fragmentiert werden müssen. Sie können also so wie sie sind abgelegt, verarbeitet und abgerufen werden. Allerdings sind die möglichen Optimierungsmaßnahmen eingeschränkt. Die Mapping-Verfahren haben den Nachteil, dass die XML-Dokumente beim Einfügen zerteilt und beim Auslesen wieder zusammengefügt respektive erzeugt werden müssen. Dadurch, dass die Daten in relationalen Tabellen vorliegen, können allerdings relationale Optimierungsmaßnahmen durchgeführt werden. Ob diese allerdings die Kosten der Fragmentierung und Rekonstruierung aufwiegen ist eine andere Frage. Ein hybrides Verfahren hätte voraussichtlich nur den Vorteil, dass insgesamt kleinere XML-Strukturen durchsucht werden müssten. Bei Abfragen müsste das XML-Dokument als solches unter Umständen auch rekonstruiert werden.
Für jedes Verfahren können und müssen natürlich im Einzelnen nach der Implementierung noch Optimierungen vorgenommen werden. Zum Beispiel ließen sich die Kosten

für den Query-Plan allein durch das Hinzufügen eines primären Index für die XML-Datentypspalte bei einer einfachen Query von über 7000 auf ca. 0.16 senken[20].

5.5 Zusammenfassung

Pauschal kann man nicht sagen, welche Strukturlösung sich am besten für welchen Sachverhalt eignet. Die Abwägungen werden nach einer Performance-Untersuchung einiger Strukturen getroffen. Die Performance-Untersuchungen aus diesem Kapitel wurden auf verschiedenen Test-Systemen durchgeführt und sind somit nicht vergleichbar. Allerdings können die in einer einzelnen Performance-Untersuchung getesteten Strukturen miteinander in Bezug gestellt werden. So erhält man zumindest eine grobe Übersicht, welche Verfahren geeignet sein könnten oder von vorne herein ungeeignet sind.
Fest steht aber, dass die strukturorientierten Mapping-Verfahren mit am schlechtesten Abschneiden. Überraschender Weise ergab eine Performanceuntersuchungen von [AEN08] über den XML-Datentyp auch schlechte Ergebnisse. Es gibt aber zwei Argumente den XML-Datentyp dennoch zu untersuchen. Erstens müssen die Querys für *NCPower* der Tamino-Semantik entsprechen, sehen also anders aus, als die getesteten. Zweitens wurden die Tests mit Microsoft SQL Server 2005 durchgeführt und nicht mit SQL Server 2008, welcher statt Tamino XML Server eingesetzt werden soll. Vor allem, da Microsoft einiges an der Implementierung des XML-Datentyps (von SQL Server 2005 zu 2008) getan hat, sind die Ergebnisse wahrscheinlich nicht mehr repräsentativ. Ein hybrides Verfahren sei im Großen und Ganzen zumindest für partiell-strukturierte XML-Daten die beste Lösung. Inwiefern das auch für die Daten von *NCPower* zutreffend ist, wird sich nach den Performance-Untersuchungen im folgenden Kapitel zeigen. Eine andere Lösung könnte genauso gut für die spezifischen Anforderungen einzelner Collections sinnvoller sein.
Das Problem ist, dass keine konkreten Ergebnisse für die Performance des nativen XML-Datentyps von SQL Server 2008 gefunden wurden. Lediglich für SQL Server 2005 wurden Untersuchungen mit partiell-strukturierten XML-Dokumenten gefunden, deren Ergebnisse aber nicht a priori auf die Anwendungsfälle von *NCPower* bezogen werden können. Die Querys von dem Redaktionssystem entsprechen einer besonderen Semantik, die in keiner der Untersuchungen verwendet wurde. Ebenso wenig konnte ein direkter Vergleich zwischen dem XML-Datentyp von SQL Server 2008 und Mapping-Verfahren gefunden werden. In Bezug auf Mapping-Verfahren ist festzuhalten, dass sie eine sehr gute Laufzeit für Querys bieten, allerdings bei der XML-Dokument Rekonstruktion sowie Speicherung schlechte Ergebnisse vorweisen.

[20] Vgl. [Col08], S. 180

Gerade bei Daten die dem offenen Inhaltsmodell folgen - explizit der Admin Collection - ist man bei der Wahl der Struktur stark eingeschränkt. Das Problem ist, dass verschieden strukturierte XML-Dokumente in einer Collection liegen. Aufgrund dieser unterschiedlichen Strukturen würde ein strukturorientiertes Verfahren unter Umständen sehr viele Tabellen erzeugen. Ebenso könnten schemaorientierte Verfahren zu unökonomischen / unperformanten Lösungen führen.
Außerdem muss bei der Wahl der Struktur einiges beachtet werden. Gegebenenfalls erzeugt ein Shredding erheblichen Overhead. Es muss von Fall zu Fall entschieden werden, ob das Verfahren so tragbar ist. Dies hängt aber von den Anforderungen und den Daten ab. Weiterhin haben Untersuchungen von [AEN08] ergeben, dass sich die Größe des Daten-Sets enorm auf die Query-Laufzeiten auswirkt. Beispielsweise verschlechtert sich bei einer bestimmten Query die Laufzeit eines bestimmten Verfahrens bei einer Verdoppelung der Datengröße sogar um über 10.000%[21]. Es muss also sehr genau darauf geachtet werden, welche Querys auf die entsprechenden Daten ausgeführt werden, um herauszufinden, welche Struktur am besten geeignet ist.
Auffällig ist auch, dass die strukturorientierten Mapping-Verfahren um die zehn Jahre alt sind. Sie werden zwar noch in aktuellen Untersuchungen verwendet, schneiden dort aber immer relativ schlecht ab. Bei den schemaorientierten Mapping-Verfahren gibt es auch aktuellere Entwicklungen, allerdings sind die Implementierungen weitaus komplexer und in der Regel nicht verfügbar.
Festzuhalten ist also, dass auf jeden Fall ein Performance-Vergleich zwischen dem nativen XML-Datentyp mit verschiedenen Indizes, einer oder mehreren hybriden Strukturen und einer relationalen Struktur durchgeführt werden muss. Der kosten-basierte Ansatz LegoDB ist eine vielversprechende Alternative, allerdings liegt dafür leider keine Implementierung vor. Um dennoch vergleichen zu können, ob sich eine relationale Struktur (also ein Mapping) nicht doch eignen könnte, kann ein einfaches Mapping manuell durchgeführt werden.

[21] Vgl. [AEN08], S. 5

6 Performance-Untersuchung

Die Strukturanalyse hat keine eindeutigen Ergebnisse sondern nur potentielle Strukturlösungen hervorgebracht. Diese Strukturen müssen in diesem Schritt so gut es geht miteinander verglichen werden um für unterschiedliche Anwendungsgebiete feststellen zu können, welche Lösung am besten geeignet ist. Dazu werden die Strukturen mit verschiedenen Querys getestet, die der Tamino-Semantik entsprechen.

6.1 Übersicht

Um ein gutes Gesamtbild von den Strukturen zu bekommen, werden verschiedene Operatoren auf diesen ausgeführt. Zu aller erst ist jedoch interessant festzustellen, wie der Speicherverbrauch aussieht. Hierbei können unterschiedliche Fälle betrachtet werden. Zum Beispiel lässt sich der Speicherverbrauch für den nativen XML-Datentyp mit und ohne Verwendung von XML Schema feststellen. Nebenbei ist die Größe der Indizes auch noch interessant.
Für die eigentlichen Performance-Untersuchungen werden mehrere verschiedene Querys (siehe unten) verwendet. Es wird versucht dadurch möglichst alle Query-Anforderungen der Collections abzudecken. Neben Querys und damit der Datenabfrage muss natürlich auch das Einfügen von Daten getestet werden. Gerade das Einfügen einer ganzen Collection könnte interessante Ergebnisse hervorbringen. Einhergehend mit dem Einfügen von Daten ist auch das Updaten für einige Collections wichtig. Das Löschen von Datensätzen ist nicht ganz so wichtig, wird aber natürlich auch untersucht.
Es wäre sehr vorteilhaft zusätzlich zu dem XML-Datentyp und verschiedener hybrider Strukturen auch ein rein relationales Mapping zu testen. Das größte Problem an der Sache ist, dass sich die schemaorientierten Verfahren performanter als die strukturorientierten herausgestellt haben. Allerdings ist die Implementierung der schemaorientierten Mappings weitaus komplexer, da diese in der Regel direkt eine ganze Middleware oder ein Framework darstellen. Es wurden drei schemaorientierte Mappings untersucht. Der DTD-Ansatz eignet sich aufgrund seiner relationalen Struktur nicht. Bei einer 19-elementigen DTD entstanden bei Tests von [KST02] 3462 Tabellen (siehe Kapitel 5.2.2.3) und das ist für *NCPower* absolut nicht tragbar. Die News-Collection ist eine der einfacher strukturierten Collections und hat allein schon ca. 90 Elemente von denen

allerdings einige redundant vorliegen. Selbst bei einer großzügigen Schätzung würden immer noch um die 40 einzigartigen Elemente übrig bleiben. Der zweite Ansatz ist die Middleware RelaXML, für welche laut [K+05] eine open source Implementierung[1] bereitgestellt ist. Allerdings sind beide Seiten nicht mehr verfügbar, sodass keine Implementierung bezogen werden kann. Der vielversprechendste Ansatz ist das kosten basierte Framework LegoDB. Es wird allerdings ebenfalls keine LegoDB Implementierung angeboten, sodass auch dieses nicht getestet werden kann.
Um dennoch eine relationale Struktur testen zu können, wird ein einfaches strukturorientiertes Mapping per Hand durchgeführt. Eine Implementierung respektive Umsetzung eines der untersuchten Verfahren würde den Workload übersteigen. Trotz eines simplen und manuell durchgeführten Mappings lässt sich erkennen, ob eine relationale Datenstruktur dem XML-Datentyp oder einem hybriden Verfahren vorzuziehen sein könnte. Sollte sich die relationale Struktur als performant erweisen, kann durch die Wahl einer besseren relationalen Struktur respektive eines der Mapping-Verfahren aus Kapitel 5, die Datenhaltung optimiert werden.
Wie im vorigen Kapitel erwähnt, gibt es mit Shrex eine Middleware, mit der man mithilfe des XML Schemas ein Mapping automatisiert erstellen kann. Dazu erweitert man das XML Schema an geeigneten Stellen um spezielle Attribute. Shrex erstellt daraufhin SQL Befehle um XML-Dokumente, die dem Schema entsprechen, an den geeigneten Stellen zu zerteilen und in die Datenbank einzufügen. Mit Shrex würden sich schnell relationale Strukturen für Untersuchungen mit anderen Collections erstellen lassen, die aus Zeitgründen nicht durchgeführt werden können. Für die verwendeten Testdaten war ein Einsatz aufgrund der Struktur der XML-Dokumente nicht erforderlich.

6.2 Performance-Tests

6.2.1 Rahmenbedingungen

Die Tests wurden auf einem Windows 7 32 Bit Rechner mit Intel Core2 6300 CPU mit 1,86GHz und 2 GB RAM durchgeführt. Durch die für die Tests nicht ganz optimalen Bedingungen können die Ergebnisse zu Durchführungszeiten auf Systemen mit besseren Spezifikationen, wie sie im Produktivbetrieb sind, stark abweichen. Eine Tendenz lässt sich im Vergleich der einzelnen Strukturen dennoch erkennen. Als Datenbank wurde Microsoft SQL Server 2008 R2 Express Edition verwendet. Diese kostenlose Version von SQL Server 2008 unterliegt einigen Einschränkungen, wie zum Beispiel die maximale Nutzung von einer CPU (beziehungsweise einem Kern) und die Unterstützung von nur 1 GB RAM.

[1] unter URL: `www.cs.aau.dk/?chr/relaxml/`.[01.02.2012] und URL: `www.relaxml.com`.[01.02.2012]

Für die Tests wurde aufgrund der Einschränkungen nicht die komplette News-Collection verwendet, sondern nur ca. die Hälfte der Daten. Dennoch haben die Testdaten eine angemessene Größe von ca. 540 MB und bestehen aus 84.062 XML-Dokumenten. Dank der Struktur können wichtige Operatoren, wie zum Beispiel die Volltextsuche, aber auch Abfragen mit Filtern auf bestimmte Attribut-Werte sowohl im Referenz- als auch im Inhalts-Bereich getestet werden.
Zuallererst wurden die Tabellen erzeugt. Sollte der XML-Datentyp verwendet werden und die Struktur die Verwendung des XML-Schemas vorsehen, so wurde vorher eine Schema Collection erzeugt. Diese wurde bei der Erzeugung der Tabelle an die XML-Datentyp Spalte gebunden. Nun wurden die nötigen Indizes festgelegt. In jedem Fall wurde für die Volltextsuche ein Volltext-Index definiert. Daraufhin konnte Datenbank beladen werden. Zusätzlich zu den anderen Performance-Tests wurde überprüft, inwiefern sich der Erstellungszeitpunkt (vor oder nach dem Beladen) des Volltext-Index Auswirkungen auf die Messergebnisse hat (siehe weiter unten). Die Query-Laufzeiten wurden in SQL Server 2008 mit SET STATISTICS gemessen:

```
SET STATISTICS TIME, IO ON
```

6.2.2 Durchführung

Es wurde versucht ein möglichst großes Spektrum an Querys zu verwenden. Das Ziel war es alle möglichen Ausprägungen, wie sie auch im Produktiv-Betrieb vorkommen zu simulieren. Aus diesem Grund wurden Querys aus dem Trace zwischen *NCPower* und Tamino als Grundlage für die Test-Querys verwendet. Die Querys lassen sich aber höchst wahrscheinlich noch optimieren um eine bessere Performance zu erzielen. Je nach Strukturart mussten die Querys umgeformt werden, da die Datenstrukturen unterschiedlich sind. Die Querys sind in logischen Gruppen zusammengefasst und durchnummeriert. Der Buchstabe hinter der Nummerierung gibt zusätzliche Informationen zu der Query an. So steht ein „k“ für die Abfrage eines kompletten Dokuments, ein „t“ steht für die Abfrage eines Teil-Dokuments und ein „s“ steht für die Sortierung. Aufgrund der verschiedenen Ausprägungen der einzelnen Querys, den verschiedenen Strukturen für die sie umgeformt werden und der Komplexität damit sie der Tamino-Semantik[2] entsprechen, finden sich die Querys auf einer CD, die dieser Arbeit beiliegt. Tabelle 6.1 gibt eine Übersicht über die bei den Tests verwendeten Querys und DML-Befehle. Querys aus der ersten logischen Gruppe (Q1.x) wählen Dokumente anhand der *ino:id* aus. Die zweite logische Gruppe (Q2.x) beinhaltet Querys mit Filtern auf den Referenz-Bereich (*BaseDocument*), wohingegen Querys aus der dritten logischen

[2]für weitere Informationen siehe „Dokumentation zum Praxisprojekt“auf der beiliegenden CD

Gruppe (Q3.x) Filter auf beide Teilbereiche nutzen. Filter auf Attribute und die Erstellungszeit beziehen sich auf den Referenz-Bereich, während im Inhalts-Bereich nur Volltextsuchen durchgeführt werden. Beim Updaten (U1) wird ein vorhandenes Dokument anhand der *ino:id* identifiziert und überschrieben. Dahingegen wird beim Einfügen (I1) lediglich ein neues Dokument eingefügt. Für das Löschen liegen zwei Fälle vor. D1 löscht ein Dokument mit bestimmter *ino:id*, D2 löscht Dokumente deren *ino:id* in einem bestimmten Wertebereich liegt. Es ist aber hinzuzufügen, dass dem Einfügen und Aktualisieren bei der relationalen Struktur noch ein Schritt vorausgeht, der nicht in der Zeitmessung enthalten ist. Aus XML-Dokumenten von *NCPower* müssen die Werte der Elemente und Attribute extrahiert werden. Bei den hybriden Strukturen finden im Vorfeld eine Fragmentierung und gegebenenfalls ebenfalls eine Extraktion der Werte statt. Analog wäre der Übersetzungsvorgang von X-Query Anfragen in SQL Querys zu nennen, der ebenfalls nicht in der Zeitmessung enthalten ist.
Um ein genaueres und eindeutigeres Ergebnis zu erhalten wird jede Query dreimal ausgeführt und die Laufzeit gemessen. Aus diesen drei Zeiten wird der Mittelwert gebildet und für die Auswertung verwendet. Die Querys dienen teilweise speziell zur Untersuchung der Laufzeit bei Abfragen von vielen Datensätzen. Im Vergleich kann man so gut sehen, wie sich die Strukturen bei unterschiedlichen Querys verhalten. So ist zum Beispiel bei Q2.2 die Ergebnismenge 64.472 Datensätze groß. Im Produktivbetrieb von *NCPower* kommt dies jedoch in der Regel nicht vor. Aus der News-Collection werden immer nur 10 oder 20 Nachrichten auf einmal benötigt, aus Editorials und Pool gegebenenfalls ein paar mehr. Lediglich aus der Admin-Collection werden beim Initialisieren von *NCPower* einmal ca. 2200 Datensätze auf einmal abgerufen.

6.2.3 Strukturen

Im Rahmen der Tests werden verschiedene Strukturen verwendet und die Ergebnisse miteinander verglichen. Es ist anzumerken, dass beinahe auf jeder Collection eine Volltextsuche durchgeführt wird. Der Volltext-Index ist sowohl für den XML-Datentyp als auch relationale Strukturen obligatorisch und somit in den Erläuterungen zu den Strukturen nicht explizit aufgeführt. Im Folgenden eine Auflistung der verschiedenen Strukturen:

XML-Datentyp

Die einfachste Struktur ist der native XML-Datentyp in SQL Server 2008. Dafür wird eine zweispaltige Tabelle verwendet, die aus einer Index-Spalte und einer XML-Datentyp-Spalte besteht. Die Index-Spalte dient zur Repräsentation der *ino:id* und die XML-

Query/DML	Semantik
Q1.1(k,t)	Abfrage eines Dokuments mit einer bestimmten *ino:id*
Q1.2(k,t,s)	Abfrage von Dokumenten, deren *ino:id* in einem bestimmten Werte-Bereich liegt
Q2.1(k,t,s)	Abfrage von Dokumenten, deren Erstellungsdatum in einem bestimmten Bereich liegt
Q2.2(k,t,s)	Abfrage mit Volltextsuche (ein Begriff) über den Referenz-Bereich
Q2.3(k,t)	Abfrage mit Volltextsuche (sechs Begriffe) über den Referenz-Bereich
Q2.4(k,t)	Abfrage von Dokumenten deren Erstellungsdatum in einem bestimmten Bereich liegt, mit Volltextsuche (sechs Begriffe) im Referenz-Bereich
Q2.5(k,t)	Abfrage eines Dokuments mit einem bestimmten Wert in einem Attribut
Q2.6(k)	Abfrage eines Dokuments mit bestimmten Werten in drei Attributen
Q3.1(k,t,s)	Abfrage von Dokumenten deren Erstellungsdatum in einem bestimmten Bereich liegt, mit Volltextsuche (sechs Begriffe) im Inhalts-Bereich
Q3.2(k,t,s)	Abfrage von Dokumente mit Volltextsuche (ein Begriff im Referenz-Bereich und sechs Begriffe im Inhalts-Bereich)
Q3.3(k)	Abfrage von Dokumenten mit bestimmten Werten in drei Attributen im Referenz-Bereich und Volltextsuche (sechs Begriffe) im Inhalts-Bereich
U1	Aktualisieren eines Dokuments mit einer bestimmten *ino:id*
I1	Einfügen eines neuen Dokuments
D1	Löschen eines Dokuments mit einer bestimmten *ino:id*
D2	Löschen von Dokumenten, deren *ino:id* in einem bestimmten Bereich liegt

Tabelle 6.1: Übersicht über Querys und DML-Befehle für die Tests

Spalte beinhaltet ein komplettes XML-Dokument (*«CompleteDocument»*). Der XML-Datentyp wird mit verschiedenen Optimierungsmaßnahmen getestet, grundlegend wird einmal mit XSD (*+mXSD*) und einmal ohne (*+oXSD*) getestet. Die verschiedenen Versionen sind: XML-Datentyp ohne Indizes (*XML*); mit primärem XML-Index (*+Prim*); mit primärem und sekundärem PATH XML-Index (*+PrimSek*). Für zwei Querys wurde zusätzlich noch der XML-Datentyp mit sekundärem PROPERTY XML-Index getestet, da dieser die Verwendung der *value()*-Funktion begünstigt. Diese Funktion wird in den Querys mit Volltextsuche verwendet, damit die Volltextsuche der Tamino-Semantik entspricht.

Hybride Strukturen

Es werden zwei verschiedene hybride Strukturen getestet, die sich im Grad der Granulierung unterscheiden. Für die XML-Datentypspalten wird ein Volltext- sowie primärer und sekundärer PATH XML-Index verwendet. Weiterhin wurde das XML Schema ebenfalls fragmentiert und mit den XML-Datentyp-Spalten verknüpft. Die erste Struktur ist grob granular (*h1*). Dabei werden die XML-Dokumente zweigeteilt. Es werden der Referenz-Bereich (*BaseDocument*) und der Inhalts-Bereich (*DocContent*) in separaten Tabellen gespeichert.
Bei der zweiten hybriden Struktur (*h2*) werden die vier Elemente von *BaseDocument*, also *DocQualifier*, *AccessRights*, *DocUserFields* und *DocSysValues*, nicht als XML-Datentyp gespeichert sondern in relationaler Form. Lediglich das Element *«DocSummary»* in *DocUserFields* wird als XML-Datentyp (mit XSD, Volltext-, primärem und sekundärem XML-Index) gespeichert. Es entstehen bei *h2* also zwei Tabellen. Die eine besteht aus Index- und XML-Datentyp-Spalte und beinhaltet den Inhalts-Bereich als XML-Datentyp. In der zweiten Tabelle wird jedes Element (das einen Wert haben könnte) und jedes Attribut als Spalte der Tabelle abgebildet. Es entsteht also für die News-Collection neben der Tabelle für den Inhalts-Bereich eine Tabelle in folgender Form:

```
newsh2(i, configName, configNumber, currentVersion,
       service, uid, updateCounter, version,
       givenDate, userDefName, AccesRights, Directory,
       Name, Status, DocSummary(XML), Title, Type, Author,
       CreationDate, LastModified, LastModifiedBy, Size)
```

Relational

Für die relationale Struktur (*rel*) wurde der Ansatz aus *h2* weitergeführt. So wurden alle Elemente (die einen Wert haben könnten) und Attribute als Spalten einer relationalen Tabelle abgebildet. Also auch der gesamte Inhalts-Bereich sowie das Element *DocSummary*. Typinformationen zu den Elementen und Attributen kann man dem XML Schema entnehmen. Für die Tests wurde jedoch aus pragmatischen Gründen hauptsächlich mit *varchar* und *int* gearbeitet. Es wird neben einem primären Index auf der Spalte *i* noch ein Volltext Index über der Spalte *AGENCY* definiert. Für alle Spalten, die in einer Volltext-Abfrage vorkommen, müsste ein Volltext-Index definiert werden. Da bei den Test-Querys nur über *AGENCY* gesucht wird, genügt das für die Tests.
Wichtig anzumerken ist hierbei zusätzlich, dass die Zusammenfassung *DocSummary*, die im Referenz-Bereich liegt, genau dem Inhalt von *document* aus dem Inhalts-Bereich entspricht. Bei der News-Collection liegen die Daten also redundant vor. Um also das

XML-Dokument nicht nur in eine relationale Struktur zu bringen, sondern auch um Optimierungen vorzunehmen, wurden die redundanten Daten weg gelassen. Für andere Collections ist das allerdings unter Umständen nicht der Fall. Der Aufbau der Tabelle sieht demnach wie folgt aus:

```
newsRel (i, configName, configNumber, currentVersion,
         service, uid, updateCounter, version,
         givenDate, userDefName, AccesRights, Directory,
         Name, Status, LastModifiedSUM, LastModifiedBySUM,
         defaultSummary, summaryName, tableSUM,
         SERVICEsum, TYPEsum, MSGNUMsum, TITLEsum,
         AGENCY, KEYWORDS, CATEGORY, WORDCOUNT, PRIORITY,
         OPTIONAL, HL1, BYTAG, LOCATION, STORYDATE,
         BODYCONTENT, TRANSDATE, BG_LIST, Title, Type,
         Author, CreationDate, LastModified,
         LastModifiedBy, Size, Attributes,
         DocumentFormat, LastModifiedCONT)
```

Beim Abrufen wird das XML-Dokument mit Hilfe des *FOR XML* Befehls wiederhergestellt. Mit Hilfe dieses Befehls ließe sich auch die *ino:id* als Attribut in das Wurzelelement der Ausgabe einfügen. So müssten die XML-Dokumente nichtmehr in einer Middleware nachmodelliert, sondern nur noch um den Tamino-Wrapper erweitert werden. Ein weiterer wichtiger Punkt ist, dass die XML-Dokumente bei diesem Mapping ohne zusätzliches Wissen nicht verlustfrei wiederhergestellt werden können. Die Struktur der Dokumente ist aus der Tabelle und den Daten allein nicht zu erfassen. Es fehlen zum Beispiel Informationen über umschließende Tags oder die Reihenfolge der Elemente. Im Falle von *NCPower* ist dies jedoch nicht so kritisch. Da sowieso eine Query-Übersetzung innerhalb einer Middleware respektive eines Gateways durchgeführt werden muss, kann an dieser Stelle die benötigte Logik und das Wissen über die Struktur ergänzt werden. Innerhalb einer Query kann also mit Hilfe des *FOR XML* Befehls die Struktur rekonstruiert werden. Der Vorteil einer solchen Aufspaltung ist, dass auf tiefere Hierarchie-Ebenen der XML-Dokumente besser zugegriffen werden kann. Ein Nachteil ist, dass das Erstellen der Querys (zum Beispiel das Übersetzen von X-Query zu SQL Querys) viele zusätzliche Informationen erfordert. Zum Beispiel falls Elemente mit dem gleichen Namen an unterschiedlichen Stellen des XML-Dokuments vorkommen und auch noch unterschiedliche Werte haben, müssen diese Identifizierbar sein. Bezogen auf die News-Collection könnte man das Element *«LastModified»* nennen. Dieses kommt dreimal an unterschiedlichen Stellen in den XML-Dokumenten vor.

6.3 Ergebnis

6.3.1 Auswertung

Die einzelnen Messergebnisse (drei pro Query) und Test-Querys (29 Querys pro Struktur) für die einzelnen Strukturen (neun insgesamt) befinden sich aufgrund ihres Umfangs auf einer beiliegenden CD. Eine tabellarische Übersicht der Mittelwerte der Messungen ist im Anhang (Abschnitt „Ergebnisse der Performance-Tests“ Tabellen A1 - A5) zu finden.
In einem stichprobenartigen Test wurde festgestellt inwiefern sich die Reihenfolge von Index-Definition und Tabellen-Beladung auf die Query-Laufzeiten auswirkt. Dafür wurde die Laufzeit einer Query (Q2.2k) für die Erzeugung des Volltext Index vor (TIB) und nach (TBI) dem Beladen der Datenbank gemessen (siehe Anhang Tabelle A5). Dabei ergab sich eine deutlich bessere Laufzeit von 123,8 Sekunden bei TIB im Vergleich zu 212,8 Sekunden bei TBI. Aus diesem Grund wurden für die weiteren Tests zuerst die Indizes definiert und danach die Tabellen beladen.
Bei dem Beladen der Strukturen wurden die Zeiten gemessen, die die Beladungen in Anspruch genommen haben. Allerdings ist in dieser Zeit ein mögliches Shredding enthalten. Es ist also nicht nur die Beladung sondern auch eine nötige Verarbeitung eingeschlossen. Zusätzlich dazu wurde nach der Beladung der benötigte Speicher (inklusive Indizes) gemessen. Das Backup aus Tamino, also die halbe News-Collection, hat eine Größe von ca. 540 MB. In SQL Server 2008 nehmen die Strukturen bis auf eine Ausnahme wesentlich mehr Speicher in Anspruch. So weisen die Strukturen *XMLoXSD*, *XMLmXSD*, *h1* und *h2* untereinander sehr ähnliche Größen auf (ca. 1.1 - 1.2 GB). Da bei der relationalen Struktur redundante Informationen weg gelassen wurden, wundert es nicht, dass diese nur ca. 260 MB Speicher verbraucht. Die Indizes (primärer, sekundärer und Volltext XML-Index) sind bei *XMLoXSD*, *XMLmXSD* und *h1* ca. 2.1 - 2.2 GB groß, bei *h2* sind es nur ca. 1.7 GB. Bei der relationalen Struktur hingegen verbrauchen die Indizes nur ca. 1.1 GB Speicher. Bei den Zeiten fällt auf, dass die grob granulare hybride Struktur *h1* bei weitem am längsten zur Beladung braucht. Die Beladung von *XMLoXSD* ist ein wenig schneller als *XMLmXSD*. Für die Beladung der hybriden Struktur *h2* und der relationalen Struktur wurde *XMLoXSD* als Grundlage genommen. Dabei konnten die XML-Fragmente beziehungsweise die Werte aus den XML-Dokumenten per *query()*- oder *value()*-Funktion ausgelesen werden.
Das Spektrum der Query-Laufzeiten ist breit gefächert. Eine nähere Betrachtung der Messergebnisse wird im Folgenden Vorgenommen. Vor allem bei den Querys, mit denen XML-Dokumente anhand der *ino:id* adressiert werden, liefern die meisten Strukturen recht gute Ergebnisse. Wobei es dabei auch Schwankungen zwischen 3,37 Sekunden

(*XMLoXSD*) und 0,25 Sekunden (*rel*) bei Q1.2k gab. Das Einfügen, Aktualisieren und Löschen hat beinahe bei allen Strukturen Laufzeiten von unter 0,5 Sekunden. Lediglich bei *XMLoXSD* dauert das Einfügen über 0,5 Sekunden und das Aktualisieren sogar knapp 2 Sekunden.
Gerade bei einer Verwendung des XML-Datentyps fällt auf, dass deine Nutzung des XML Schemas auf jeden Fall zu bevorzugen ist. In Abbildung 6.1 sieht man die unterschiedlichen Ausprägungen der Laufzeiten für vier ausgewählte Querys bei den Strukturen *XMLoXSD* und *XMLmXSD*. So sind bei den Querys Q2.2k und Q2.5k die Laufzeiten beinahe gleich. Einen großen Unterschied gibt es bei Q2.3k. Bei dieser Query liefert die Struktur *XMLmXSD* eine etwa 2,8-mal bessere Laufzeit.

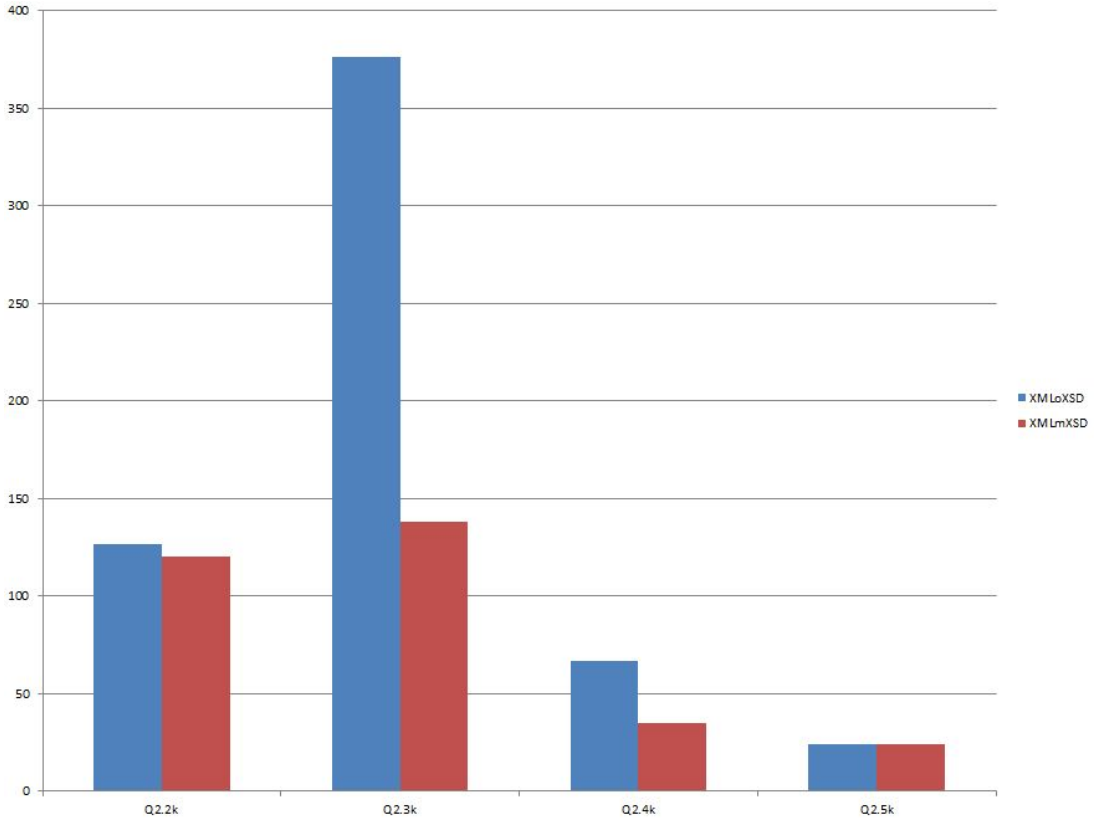


Abbildung 6.1: Laufzeitenvergleich für die Strukturen XMLoXSD und XMLmXSD

Vergleicht man die verschiedenen Möglichkeiten den XML-Datentyp zu verwenden, so bietet im Durchschnitt die Version mit XML Schema, aber ohne Indizes (*XMLmXSD*), die besten Laufzeiten. Bei einzelnen Querys sind die Versionen mit primärem oder beiden XML-Indizes allerdings besser. Das Gesamtbild der Laufzeiten bei *XMLmXSD* ist recht konstant. Dadurch wird diese Struktur zu einer guten Lösung, falls die Anforderungen an die Struktur sehr vielseitig sind.

Sollten Filter auf Attribut-Werte in den Querys auftreten (so wie bei Q2.5, Q2.6 und Q3.3) liefern die Strukturen mit primärem und sekundärem PATH XML-Index hervorragende Laufzeiten. Tabelle 6.2 zeigt die Laufzeiten für die drei genannten Querys bei den Strukturen *XMLmXSDPrimSek*, *h1*, *h2* und *rel*. *XMLmXSDPrimSek* liefert für die drei genannten Querys die besten Laufzeiten. Die feiner granulierte hybride Struktur *h2* liefert die schlechtesten Ergebnisse. Sollte nur der primäre XML-Index für den XML-Datentyp verwendet werden, sind die Laufzeiten sogar noch schlechter als bei der Verwendung von gar keinem Index.

Query	**XMLmXSDPrimSek**	**h1**	**h2**	**rel**
Q2.5k	0,07133333	0,21933333	7,795	0,21166667
Q2.5t	0,05933333	0,20366667	2,70966667	0,15133333
Q2.6k	0,01766667	0,22	0,54166667	0,21066667
Q3.3k	0,23233333	4,023	6,775	0,49

Tabelle 6.2: Laufzeiten für Querys mit Filtern auf Attribut-Werte

Da in den Querys mit Volltextsuche aufgrund der Tamino-Semantik die *value()*-Funktion verwendet werden muss, könnte der sekundäre PROPERTY XML Index zur Optimierung dienen. Die Querys Q2.2k und Q3.2k hatten aber trotz des zusätzlichen Indexes keine besseren Laufzeiten, sondern liefen sogar schlechter. Die Gründe dafür sind weiter unten erläutert.

Bei der Betrachtung der Messergebnisse fällt ganz besonders auf, dass die Verwendung der Volltextsuche über den XML-Datentyp zu sehr schlechten Laufzeiten führt. Abbildung 6.2 zeigt die Laufzeiten für die Q2.2k, Q2.3k und Q3.2k, die alle die Volltextsuche verwenden. Dabei werden die Strukturen *XMLmXSD*, *XMLmXSDPrimSek*, *h1*, *h2* und *rel* miteinander verglichen. Die beiden hybriden Strukturen haben sehr schlechte Laufzeiten. Das liegt zum Teil daran, dass bei den hybriden Strukturen zwei Tabellen gejoined werden, was zu einer höheren Laufzeit führt. An dieser Stelle sei nochmals erwähnt, dass bei den Test-Querys teilweise sehr viele Datensätze abgerufen. Die große Anzahl diente lediglich zur Performance-Überprüfung, im Produktiv-Betrieb werden weitaus weniger Datensätze auf einmal abgerufen. Die generellen Gründe für die schlechte Laufzeit der Querys mit Volltextsuche werden in Kapitel 6.3.2 erläutert.

Die Querys über die relationale Struktur haben sehr gute Laufzeiten, sei es mit oder ohne Volltextsuche. Sollte allerdings die Volltextsuche über einen größeren Bereich des XML-Dokuments durchgeführt werden, würde in der relationalen Struktur über mehrere Spalten gesucht. Um zu überprüfen, wie sich die relationale Struktur bei so einer Volltextsuche verhält, wurden zwei weitere Querys erstellt. Dafür muss der Volltext-Index die Spalte *AGENCY*, *TITLEsum* und *CATEGORY* enthalten. Die erste Query

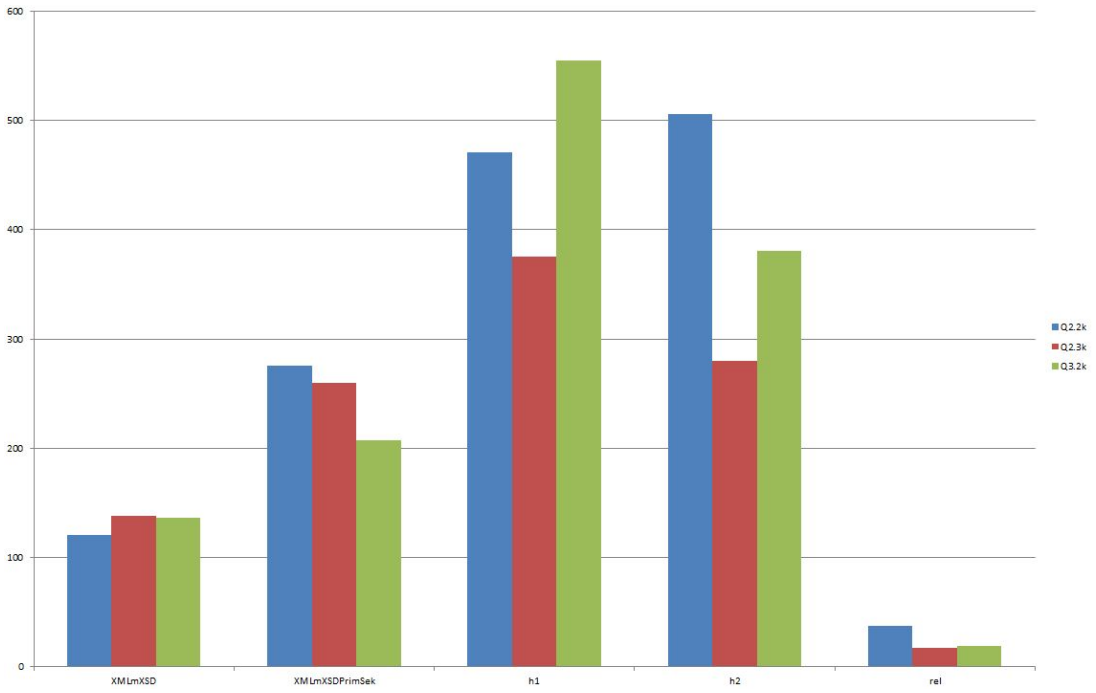


Abbildung 6.2: Laufzeitenvergleich bei der Volltextsuche

liefert einen Datensatz bei einer Volltextsuche über alle drei Spalten (Z1) und die zweite Query liefert 12.701 Datensätze bei einer Volltextsuche über zwei Spalten (Z2). Z1 hat eine Laufzeit von 0,89 und Z2 von 4,05 Sekunden.
Die relationale Struktur liefert mit Abstand die besten Laufzeiten. Betrachtet man die durchschnittliche Laufzeit aller Querys, so kommt man auf einen Wert von ca. 5,11 Sekunden. Von der durchschnittlichen Laufzeit her gesehen die zweitbeste Struktur *XMLmXSD* hat einen Wert von 46,2 Sekunden. Die Struktur mit den längsten Laufzeiten ist *h1* mit einem Durchschnitt von 121,15 Sekunden. Allerdings ist für die relationale Struktur die meiste Implementierungsarbeit nötig. Unter anderem ist die Übersetzung aufwändiger als bei den anderen Strukturen, da von einer Abfrage über XML-Dokumente in eine Abfrage über Relationen übersetzt werden muss. Bei der Übersetzung ist ganz besonders auf die Einhaltung der Semantik von Volltextsuchen zu achten. Während den Tests hat sich herausgestellt, dass die geradlinige Übersetzung der Test-Querys bei der relationalen Struktur teilweise eine abweichende Anzahl an Datensätzen lieferte. Bei Q2.3 wurden 56.150 statt 45.394, bei Q2.4 154 statt 179, bei Q3.1 ebenfalls 154 statt 179 und bei Q3.2 56.150 statt 34.908 Datensätze geliefert. Die Übersetzung der Querys[3] muss je nach Struktur angepasst werden. Korrekturen und Optimierungen können in diesem Schritt durchgeführt werden.

[3]Bisher wurde die Übersetzung von X-Query zu SQL Querys für den XML-Datentyp entwickelt

6.3.2 Anmerkungen

Zu den Performance-Untersuchungen ist noch etwas Wichtiges hinzuzufügen. Generell ist die Volltextsuche über dem XML-Datentyp in SQL Server 2008 sehr performant. In den Untersuchungen sind die langen Laufzeiten der Querys mit Volltextsuche auf die Abbildung der Tamino-Semantik zurückzuführen. Dazu gehört die mehrfache Verwendung verschiedener Funktionen für den XML-Datentyp. Dies wurde durch einen kleinen Test auch nochmals belegt. Eine einfache Volltextsuche über der XML-Datentypspalte mit dem SQL Operator CONTAINS und ohne die zusätzlichen XML-Funktionen hatte sehr gute Laufzeiten (siehe Anhang Tabelle A5). Dabei wurde nur die in einer separaten Spalte gespeicherte *ino:id* abgerufen. Bei einer Suche nach einem Wort und einer Ergebnismenge von 64.613 Datensätzen brauchte die Query ca. 1,11 Sekunden. Bei einer Suche nach sechs Wörtern und einer Ergebnismenge von 72.885 Datensätzen betrug die Laufzeit ca. 1,21 Sekunden. Wird allerdings ein XML-Dokument abgerufen, so erhöht sich die Laufzeit drastisch. Die Suche nach einem Wort hat, bei einer Abfrage der XML-Dokumente anstatt der *ino:id* eine Laufzeit von 153,55 Sekunden mit einer Ergebnismenge von ebenfalls 64.613 Datensätzen. Bei den Querys entsprach die Ergebnismenge natürlich nicht derselben, wie bei den Querys nach Tamino-Semantik.
Die Abbildung der Tamino-Semantik bei den Querys ist auf jeden Fall zu optimieren, um weitaus bessere Ergebnisse zu erzielen. Dies kann durch umformulieren der Querys erreicht werden. Zum Beispiel könnte man durch die Volltextsuche die Ergebnismenge einschränken und daraufhin durch weitere Bedingungen die Semantik von Tamino abbilden. Inwiefern sich der Einsatz der XML-Funktionen optimieren lässt, bleibt an dieser Stelle offen. Ebenfalls könnte man die Menge der abzurufenden Datensätze einschränken. Die Query Q3.2k (mit Tamino-Semantik), die bei der Struktur *XMLmXSD* eine Laufzeit von ca. 135,84 Sekunden hat, braucht lediglich ca. 1,10 Sekunden um die ersten zehn Datensätze abzurufen (Siehe Anhang Tabelle A5 - TOP 10 XMLmXSD).
Da dieser Sachverhalt erst zum Ende der Arbeit auffiel, blieb nicht genügend Zeit, umfassende Vergleiche zwischen optimierten Querys über den XML-Datentyp und der relationalen Struktur durchzuführen. Im Nachhinein wären zusätzliche Test-Querys zum generellen Testen der Volltextsuche bei den Strukturen sinnvoll gewesen. So hätte sich deutlicher gezeigt, wie sich die Laufzeiten bei normalen und bei den Querys mit Tamino-Semantik für die einzelnen Strukturen unterscheiden.

7 Resultat für das Redaktionssystem

In den vorangegangen Kapiteln wurden sowohl das Redaktionssystem *NCPower*, die Datenbanken Tamino XML Server und Microsoft SQL Server 2008, als auch verschiedene Strukturalternativen für die Speicherung von XML-Strukturen in relationalen Datenbanken untersucht. Zusätzlich wurden verschiedene Strukturen anhand von Querys, wie sie im Produktiv-Betrieb vorkommen könnten, getestet. Die verschiedenen Anforderungen, Eigenschaften und Voraussetzungen ergeben letztlich die Abbildung der XML-Strukturen in SQL Server 2008 für die einzelnen Collections.

7.1 Abwägungen

Unabhängig von den Collections besteht die Anforderung, dass die abgefragte XML-Struktur im Wurzel-Element die *ino:id* als Attribut enthält. Dieses Attribut ist nicht persistent im XML-Dokument gespeichert, sondern wird von Tamino an das Wurzel-Element der Abgefragten XML-Struktur gehängt. Auf diese *ino:id* können aber dennoch Abfragen ausgeführt werden. SQL Server 2008 muss also auch die Möglichkeit bieten Abfragen auf das Attribut zu simulieren und dieses an die Ausgabe zu hängen. Die *ino:id* könnte als Index-Spalte in SQL Server dargestellt werden. Abfragen auf das Attribut sind dadurch möglich und auch die eindeutige Identifizierung ist gewährleistet. Eine Speicherung in den XML-Dokumenten, sollten diese als XML-Datentyp gespeichert werden, ist nicht sinnvoll, da im Prinzip jedes Element das Wurzel-Element der Ausgabe sein könnte. Es gibt zwei Möglichkeiten die *ino:id* als Attribut in das Wurzel-Element der abgefragten XML-Struktur zu bringen. Zum einen kann dies in SQL Server oder aber im Gateway geschehen. In SQL Server müsste man „FOR XML“ verwenden und in dem Gateway bietet sich die Nutzung von DOM an. Eine Einbindung mit „FOR XML“ist problemlos und performant, wie sich bei der Rekonstruktion kompletter XML-Dokumente aus relationalen Daten in Kapitel 6 gezeigt hat. Marcus Paradies ([P$^+$10]) untersucht die Performance von der XML Verarbeitung in der Datenbank (DB2 9.7) und einer Middleware. Da die Verarbeitung in der Datenbank aber zu signifikant besseren Ergebnissen geführt hat, kann man davon ausgehen, dass die Ergebnisse mit SQL Server 2008 ähnlich wären. Die *ino:id* sollte also in der Datenbank an das Wurzelelement der abgerufenen XML-Dokumente gehängt werden.

> „In use case 1, extracting values from XML documents and assigning them to Java objects is up to 3x faster when the extraction happens with SQL/XML rather than SAX. In use case 2, splitting large XML documents into smaller XML fragments with SQL/XML is up to 3x faster than with DOM-based implementation. In use case 3, modifying existing XML documents with SQL/XML and XQuery Updates in the database is 2.2x to 2.6x faster than performing the equivalent operations with DOM in middle-tier code. At the same time, the SQL/XML implementations require 10x to 13x less code than the Java-based middle-tier implementations.“ ([P+10], S. 4)

Gegebenenfalls bietet sich auch die Möglichkeit den Tamino Wrapper, welcher das ResultSet von Tamino enthält, in SQL Server 2008 zu modellieren. Das hätte den Vorteil, dass die Verarbeitung der XML-Dokumente beinahe vollständig auf die Datenbank verlagert werden würde. So müssten die XML-Dokumente nur noch durch das Gateway „geschleust“ werden, was ebenfalls Performance-Vorteile mit sich bringen würde. Eine genauere Untersuchung inwiefern eine Verarbeitung der Tamino Wrapper möglich ist, muss allerdings separat untersucht werden.
Ein Mapping-Verfahren müssten in der Middleware realisiert werden. Dies gilt vor allem für schemaorientierte Mappings, da hier eine Menge zusätzlicher Logik benötigt wird. Generell muss das Einfügen beziehungsweise Speichern von Dokumenten in der Middleware übersetzt werden. *NCPower* verschickt komplette XML-Dokumente unter dem Parameter *_process* per http. Sollte das XML-Dokument relational gespeichert werden, so müssten zum Beispiel erst die Werte der Elemente und Attribute extrahiert werden. Da aber sowieso eine Middleware respektive das Gateway benötigt wird, um X-Query Anfragen in SQL Querys zu übersetzen, können an dieser Stelle auch noch Erweiterungen vorgenommen werden.
Eine Verwendung von XSD hat den großen Vorteil, dass dadurch Performance-Vorteile entstehen. Da sich die Struktur der XML-Dokumente nicht verändert, verändert sich auch das XML Schema nicht. Es gibt also weder ein Schema Chaos (jede Collection hat genau ein XML Schema), noch eine Schema Evolution (die Schemata / Struktur der XML-Dokumente ändert sich nicht). Der XML-Datentyp sollte also nach Möglichkeit mit dem entsprechenden XML Schema verwendet werden.
Sollten die XML-Dokumente in einer relationalen Struktur abgelegt werden, so müssen diese nicht zwingend verlustfrei gespeichert werden. Die Logik zum Rekonstruieren der Struktur kann auch in der Middleware respektive dem Gateway liegen. Da die Tamino X-Query Anfragen sowieso in SQL Querys übersetzt werden müssen, kann hier die Struktur mit Hilfe der „FOR XML“-Befehle nachgebildet werden. Durch diesen Informationsverlust entsteht zusätzlich ein Speicherplatzvorteil, der alleine jedoch

kein Argument für eine verlustbehaftete Speicherung ist. Inwiefern sich dies auf XML-Dokumente von *NCPower* anwenden lässt, hängt von der entsprechenden Collection ab.

7.2 Kriterien für die Strukturen

Der XML-Datentyp ist universell einsetzbar, solange die XML-Dokumente wohlgeformt sind. Dabei spielt es keine Rolle ob XML Schema verwendet wird. Allerdings lassen sich mit ein paar Untersuchungen Optimierungsmöglichkeiten oder effizientere Strukturen finden. Es sei aber angemerkt, dass die Kriterien in diesem Kapitel nur als Heuristiken, nicht aber als verbindliche Regeln anzusehen sind. Folgende Fragen helfen dabei eine Struktur auszuwählen:

- XML Schema
  - Gibt es ein XML-Schema?
  - Welchem Inhaltsmodell folgen die XML-Dokumente?
  - Wie viele verschiedene Ausprägungen haben die XML-Dokumente?
  - Welche strukturellen Besonderheiten weisen die XML-Dokumente auf?
- Querys
  - Gibt es Volltextsuchen?
  - Gibt es Querys mit Filtern auf Attribut-Werte?
  - Werden komplette und / oder Teil-Dokumente abgefragt?
  - Wie groß ist die Ergebnismenge?
- Zugriffe
  - Gibt es mehr lesende oder schreibende Zugriffe?

Das Vorhandensein und der Aufbau des XML Schemas schränken die Wahl der Struktur ein. Sollte das Schema das offene Inhaltsmodell vorgeben oder gibt es gar kein XML Schema, dann kann auch kein schemaorientiertes Mapping angewandt werden. Selbst strukturorientierte Mappings sollten dann aufgrund der Vielzahl möglicher Ausprägungen der XML-Dokumente vermieden werden. Der XML-Datentyp ist in diesem Fall wahrscheinlich die beste Lösung. Gibt das XML Schema aber viele optionale Elemente, Wiederholungsgruppen und verschiedene mögliche Ausprägungen (durch das Element *«choice»*) vor, so ist neben dem XML-Datentyp trotzdem noch ein schemaorientiertes Mapping möglich. Strukturorientierte Mapping-Verfahren würden bei solchen

XML-Strukturen wahrscheinlich zu schlechten Ergebnissen führen. Sollte das Schema die Struktur der XML-Dokumente stark einschränken (keine Wiederholungsgruppen, keine optionalen Elemente und nur eine mögliche Ausprägung), so bietet sich auch ein strukturorientiertes Mapping an.
Ein weiterer Punkt der bei der Wahl der Struktur eine Rolle spielt, ist die Art und Weise, wie die Querys aussehen. Gibt es Filter-Bedingungen auf Attribut-Werte, so eignet sich der XML-Datentyp mit primärem und sekundärem PATH Index sehr gut. Bei kleineren Ergebnismengen kann generell der XML-Datentyp in Betracht gezogen werden. Sollte die Ergebnismenge jedoch sehr groß sein, bietet eine relationale Struktur wahrscheinlich eine bessere Performance. Bei Volltextsuchen ist ein Volltext-Index sowohl für den XML-Datentyp, als auch relationale Strukturen erforderlich.
Ebenfalls wichtig ist die Frage, ob es eher viele lesende oder schreibende Zugriffe gibt. Werden kaum Daten abgefragt (wie bei der Archive-Collection) dann ist unter Umständen der XML-Datentyp eine gute Wahl. Bei vielen lesenden Zugriffen sollte auf die Art der Querys (siehe oben) geachtet werden.

7.3 Strukturergebnisse für die Collections

Für die einzelnen Collections ergeben sich durch die Unterschiedlichen Anforderungen und Schemata unterschiedliche Strukturlösungen. Die Überprüfung der Ergebnisse aus der Strukturanalyse in Kapitel 6 hat ergeben, dass eine relationale Speicherung durchschnittlich am performantesten ist. Allerdings sind die Querys für den XML-Datentyp noch stark optimierbar. Allen Collections ist definitiv die Behandlung der *ino:id* gemeinsam. Eine Speicherung in einer separaten Index-Spalte in SQL Server 2008 bietet sich an. Im Folgenden wird auf die einzelnen Collections genauer eingegangen um eine passende Struktur für die entsprechenden Daten zu finden.

Admin

Bei der Admin-Collection ist das größte Problem, dass diese dem offenen Inhaltsmodell folgt. Das Schema gibt hierbei nur den groben Aufbau des XML-Dokuments vor. Dabei handelt es sich um die Zweiteilung in Referenz- (*BaseDocument*) und Inhalts-Bereich (*DocContent*), sowie optionale Elemente für den Inhalt. Untersuchungen der Collection haben aber mindestens 25 verschiedene Ausprägungen des Inhalts-Bereichs hervorgebracht. SQL Server bietet keine Möglichkeit für das offene Inhaltsmodell an. Das heißt also, dass jegliches XML-Dokument aus der Admin-Collection nicht gegen das Schema validieren würde. Es kann also nicht verwendet werden. Zusätzlich sind durch das offene Inhaltsmodell sowohl struktur- als auch schemaorientierte Mapping-Verfahren sehr schwierig umzusetzen. Aufgrund der Tatsache, dass Querys an die Collection (siehe

Anhang) sehr häufig Filter auf Attribut-Werte haben, kann der XML-Datentyp mit primärem und sekundärem PATH XML-Index verwendet werden. Diese Struktur bietet auch ohne Verwendung von XSD bei reinen Attribut-Filtern sehr gute Laufzeiten. Die Admin-Collection ist eine der kleineren Collections (6540 Datensätze, ca. 50 MB), wodurch die Verwendung des XML-Datentyps generell begünstigt ist.
Strukturlösung: XML-Datentyp ohne XSD aber mit primärem und sekundärem PATH XML-Index (*XMLoXSDPrimSek*)

Archive

Für die Archive-Collection gibt es kaum lesende Zugriffe, allerdings werden Daten regelmäßig weggeschrieben. Solange das Wegschreiben nicht den Betrieb von *NCPower* einschränkt, ist die Performance nicht ganz so wichtig wie bei anderen Collections. Ein Problem welches bei der Collection auftritt ist, dass der Inhalt entweder ein Skript oder ein Rundown ist. Weiterhin unterscheiden sich *DocSummary* und *Document*, also die Zusammenfassung und das eigentliche Dokument. Zusätzlich kommen viele optionale Wiederholungsgruppe innerhalb der eigentlichen Dokumente vor (siehe Kapitel 4.3.3). Dies sind alles Gründe, die gegen eine relationale Struktur sprechen. Da es kaum lesende Zugriffe gibt und lediglich Daten in regelmäßigen Abständen gespeichert werden bietet sich der XML-Datentyp an. In den Querys auf die Collection (siehe im Anhang die Querys zu Editorials) treten oftmals Filter auf Attribute und Element-Werte auf, dadurch bietet sich zusätzlich die Verwendung der beiden XML Indizes an.
Strukturlösung: XML-Datentyp mit XSD und primärem sowie sekundärem PATH XML-Index (*XMLmXSDPrimSek*)

CMS

CMS beinhaltet multimediale Inhalte und ist die größte Collection[1]. Da beim Fernsehen viel mit multimedialen Inhalten gearbeitet wird, gibt es entsprechend viele lesende und schreibende Zugriffe. Das eigentliche Dokument kann sechs verschiedene Ausprägungen haben. Es werden sowohl komplette als auch Teil-Dokumente abgefragt. Zusätzlich werden in Querys sowohl im Referenz- als auch im Inhalts-Bereich nach bestimmten Werten gesucht. Wobei im Inhalts-Bereich zusätzlich viele Volltextsuchen durchgeführt werden. Aufgrund der vielen Volltextsuchen würde sich eine relationale Struktur anbieten, es sei denn, eine Optimierung der Querys führt zu einer ähnlich guten Performance für den XML-Datentyp. Da in dem XML Schema zu der CMS-Collection mehrere optionale Elemente und mehrere optionale Wiederholungsgruppen vorkommen, muss eine Struktur gewählt werden, die diese Besonderheiten handhaben kann. Sollte der Einsatz eines schemaorientierten Mapping-Verfahrens nicht möglich sein, kann die Struktur *XML-*

[1] ein Backup der Collection aus dem Testsystem vom 01.09.2011 war ca. 5.5 GB groß

mXSDPrimSek verwendet werden. Die Test-Query Q3.1 spiegelt die typische Abfrage auf die CMS-Collection von der Semantik her wieder, auch wenn aufgrund einer anderen Test-Collection nach anderen Werten gesucht wird. Die Laufzeiten von Q3.1 liegen bei der Struktur *XMLmXSDPrimSek* bei unter 1,5 Sekunden bei einer Ergebnismenge von ca. 180 Datensätzen.
Strukturlösung: Schemaorientiertes Mapping-Verfahren oder XML-Datentyp mit XSD und primärem sowie sekundärem PATH XML-Index (*XMLmXSDPrimSek*)

Editorials (inkl. Versioning)

Eine der wichtigsten Collections ist Editorials mit zugehöriger Versionierung. In dieser Collection liegen die Sendepläne und Skripte. Gerade Sendepläne werden häufig aktualisiert und mehrere Nutzer arbeiten gleichzeitig daran. Es gibt also häufige Lese- und Schreib-Zugriffe. Dabei wird sowohl das komplette als auch Teil-Dokumente abgerufen. Querys greifen auf den Referenz- und den Inhalts-Bereich zu. Im Referenz-Bereich wird entweder auf Attribute oder Elemente gefiltert, im Inhalts-Bereich gibt es auch Volltextsuchen. Die Querys entsprechen also den Test-Querys Q2.5, Q2.6, Q3.1 und Q3.3. Aufgrund der Besonderheit beim Aktualisieren von Daten in Tamino (mit dem _process Parameter) werden vorhandene Dokumente einfach überschrieben. Außerdem ist diese Collection sehr umfangreich und beinhaltet viele Daten[2]. Es gibt mehrere Optionale Elemente und optionale Wiederholungsgruppen. Ein schemaorientiertes Mapping würde wahrscheinlich die beste Performance bringen. Da es aber vor allem in den Sendeplänen viele Wiederholungsgruppen (zum Beispiel die einzelnen Zeilen des Sendeplans) gibt, könnte es dabei zu Komplikationen kommen.
Strukturlösung: Schemaorientiertes Mapping-Verfahren oder XML-Datentyp mit XSD und primärem sowie sekundärem PATH XML-Index (*XMLmXSDPrimSek*)

Messaging

Die Messaging-Collection wird nicht sehr häufig benutzt. Suchen innerhalb der Collection laufen sowohl über den Referenz- als auch über den Inhalts-Bereich. Daten werden gelegentlich gespeichert und abgerufen. Hinzu kommt ein relativ kleiner Datenbestand. Für diese Collection eignet sich der native XML-Datentyp oder aufgrund der einfachen Struktur eine Adaption der relationalen Struktur aus Kapitel 6. Genau wie bei der CMS-Collection spiegelt die Test-Query Q3.1 eine typische Abfrage wieder.
Strukturlösung: Adaption der relationalen Struktur aus Kapitel 6 oder XML-Datentyp mit XSD und primärem sowie sekundärem PATH XML-Index (*XMLmXSDPrimSek*)

[2] ein Backup der Collection aus dem Testsystem vom 01.09.2011 war über 3 GB groß

News

Die News-Collection wird hochfrequentiert genutzt. Jede Minute kommen Agenturmeldungen in das System, welche in der Collection gespeichert werden. Zusätzlich recherchieren Redakteure über den Meldungen. Es werden die aktuellsten Meldungen abgerufen oder es wird nach bestimmten Stichworten in den Meldungen gesucht. Wegen der Volltextsuche über dem Inhalts-Bereich oder auch der Suche nach bestimmten Kriterien im Referenz-Bereich, würde sich ein Mapping anbieten. Es wird sehr häufig der gesamte Referenz-Bereich abgerufen, was den Rekonstruktionsaufwand verringert. Allerdings werden auch andauernd neue Dokumente in die Datenbank geschrieben. Der Performance-Verlust beim Übersetzen des Tamino *_process*-Parameter in ein SQL INSERT müsste noch untersucht werden. Durch die eindeutige und geradlinige Struktur der XML-Dokumente (keine Rekursionen) können diese gut in eine relationale Struktur gebracht werden.
Strukturlösung: relationale Struktur aus Kapitel 6

Pool (inkl. Versioning)

Die Pool- und Pool.Versioning-Collection sind fast identisch und dienen als „Ablage“ inklusive Versionierung für die Nutzer. Die Collection umfasst Agenturmeldungen und Skripte. Sobald etwas in Pool gespeichert wird, wird es auch in Pool.Versioning geschrieben. Für das Lesen gilt dies allerdings nicht. Aus der Pool.Versioning werden nur Daten abgerufen, wenn der Nutzer eine ältere Version von einem Dokument abrufen will. Da in dem Trace zwischen *NCPower* und Tamino keine Querys gefunden wurde, kann man keine genaue Aussage über die Datenabfrage treffen. Es ist aber anzunehmen, dass bei Querys nach dem Besitzer und gegebenenfalls dem Typ des Dokuments (Skript oder Agenturmeldung) gefiltert wird. Im XML Schema zur Pool-Collection kommen mehrere optionale Elemente und optionale Wiederholungsgruppen vor. Für die Speicherung der Agenturmeldungen würde sich die relationale Struktur anbieten (siehe oben). Da sich die Struktur von den Skripten im Gegensatz zu den Agenturmeldungen zwischen Zusammenfassung (*DocSummary*) und Dokument (*Document*) unterscheidet, ist eine manuelle Implementierung einer relationalen Struktur dafür aufwändiger. Ein schemaorientiertes Mapping wäre am geeignetsten. Wenn man davon ausgehen könnte, dass es keine Volltextsuchen im Referenz- und Inhalts-Bereich gleichzeitig gibt (oder eine Query-Optimierung gute Ergebnisse hervorbringt), würde sich *XMLmXSDPrimSek* anbieten.
Strukturlösung: Schemaorientiertes Mapping-Verfahren oder XML-Datentyp mit XSD und primärem sowie sekundärem PATH XML-Index (*XMLmXSDPrimSek*)

7.4 Exkurs: Auswirkungen auf das Gateway

Sollten die Strukturen so eingesetzt werden, wie in diesem Kapitel erläutert, so kommt es zu einigen Auswirkungen auf das Gateway. Die alleinige Verwendung des XML-Datentyps in SQL Server 2008 ist bereits im Gateway implementiert. Für die relationalen Strukturen einiger Collections müssten allerdings Erweiterungen vorgenommen werden.
An erster Stelle heißt das, dass Querys je nachdem an welche Collection sie gerichtet sind, anders übersetzt werden müssten. Für jede Struktur muss also ein separater Übersetzer entwickelt werden. Grundlegend kann man dabei vom Aufbau des Übersetzers aus dem Gateway respektive des X-Query Parsers und Lexers aus dem vorangegangenen Projekt ausgehen. Weiterhin bieten die Test-Querys, die dieser Arbeit beiliegen, eine gute Grundlage um Abfragen auf relationale Strukturen in dem Gateway zu implementieren. Vor allem die Rekonstruktion der XML-Dokumente unter Verwendung von *FOR XML* kann einfach adaptiert werden.
Sollte eine Implementierung des Tamino Wrappers in SQL Server 2008 möglich und sinnvoll sein, so kann man die komplette Verarbeitung des Ergebnisses in die Datenbank verlagern. So müsste das Ergebnis-Dokument nur noch durch das Gateway zurück an *NCPower* geschleust werden. Ein hinzufügen der *ino:id* in der Datenbank ist in jedem Fall sinnvoll, da diese als Wert in der Index-Spalte gespeichert wird. So müssen die Dokumente nicht erneut im Gateway verarbeitet werden, wie es bisher der Fall ist.

8 Fazit

Bei den Untersuchungen hat sich wie erwartet herausgestellt, dass sich für die unterschiedlichen Collections unterschiedliche Strukturen eignen. Denn nicht nur die Anforderungen an die einzelnen Collections führen zu anderen Strukturen, sondern auch die Struktur der XML-Dokumente an sich. Die Strukturlösungen reichen vom XML-Datentyp mit verschiedenen Indizes, zu rein relationalen Strukturen. Allerdings müsste die Performance der schemaorientierten Mapping-Verfahren noch untersucht werden.
Vor allem die getestete relationale Struktur hat sich als äußerst effizient herausgestellt. Aus pragmatischen Gründen - es lag keine Implementierung der untersuchten Mapping-Verfahren vor und eine Umsetzung dieser hätte den Workload überstiegen - wurde allerdings nur ein einfaches strukturorientiertes Mapping getestet. Die Operatoren wurden dafür manuell übersetzt und die Struktur wurde an die zugehörige Collection angepasst. So sind zum Beispiel redundante Daten ausgelassen worden. Dennoch gibt es trotz der sehr guten Query-Laufzeiten noch Optimierungspotential bei der relationalen Struktur. Bei den Performance-Untersuchungen hat sich zusätzlich herausgestellt, dass die hybriden Strukturen und damit die fragmentierte Speicherung der XML-Dokumente nicht sehr effizient sind. Im Gegensatz dazu ist das Rekonstruieren / Erzeugen von XML-Dokumenten aus relationalen Daten mit *FOR XML* sehr performant. Es ist auf eine ausreichende Spezifikation des Systems zu achten, sollte der XML-Datentyp verwendet werden. Dieser nimmt bei einer Verarbeitung verhältnismäßig viel Leistung und Speicher in Anspruch.
Performance-Tests mit den schemaorientierten Mapping-Verfahren, genauso wie umfassende Lasttests, konnten im Rahmen dieser Arbeit leider nicht durchgeführt werden. Zusätzlich lag zuerst die Vermutung nahe, dass die Volltextsuche über den XML-Datentyp zu einer sehr schlechten Performance führte. Wie aber in zusätzlichen Tests festgestellt wurde, lag die schlechte Laufzeit der Querys an der Abbildung der Tamino-Semantik. Dafür mussten mehrfach Funktionen für den XML-Datentyp verwendet werden, welche zu der schlechten Laufzeit der Querys führten. Die Querys mit der Tamino–Semantik müssen bei einer Verwendung des XML-Datentyps definitiv optimiert werden.

Ausblick

In der Kürze der Zeit, ergaben sich weitere offene Fragen. Gerade bei der Verwendung von relationalen Strukturen muss noch untersucht werden, wie groß der Performance-Verlust durch das Zerteilen der XML-Dokumente ist. Zudem würde sich bei einigen Collections voraussichtlich ein schemaorientiertes Mapping als Struktur eignen. Um dies zu verifizieren müssten Performance-Untersuchungen mit diesen Strukturen durchgeführt werden. Bisher war dies nicht möglich, da keine Implementierungen verfügbar waren und eine Umsetzung zu viel Zeit in Anspruch genommen hätte. Zusätzlich bietet sich eine Anpassung der relationalen Struktur an die Anforderungen der Collection an, für die sie verwendet wird. Eine Evaluation der Strukturlösungen durch Lasttests auf Systemen mit einer besseren Spezifikation würde zeigen, ob die Strukturen tatsächlich den hohen Anforderungen des Redaktionssystems genügen. Hierbei werden viele gleichzeitige Zugriffe auf die Datenbank, wie sie im Produktiv-Betrieb vorkommen würden, simuliert. Letztendlich ist eine Weiterentwicklung des Gateways und damit verbunden der Einsatz von *NCPower* mit Microsoft SQL Server 2008 das zu erreichende Ziel.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

h1	grob granulare hybride Struktur bestehend aus zwei Teilen (Referenz- und Inhaltsbereich), die jeweils als XML-Datentyp gespeichert werden.
h2	feiner granulare hybride Struktur bestehend aus zwei Teilen (Referenz- und Inhaltsbereich). Der Inhalts-Bereich wird als XML-Datentyp, der Referenz-Bereich in relationaler Form gespeichert.
rel	relationale Struktur, bei der redundante Informationen weg gelassen werden.
XMLmXSD	XML-Datentyp mit XML Schema.
XMLmXSDPrim	XML-Datentyp mit XML Schema und primärem XML-Index.
XMLmXSDPrimSek	XML-Datentyp mit XML Schema und primärem sowie sekundärem PATH XML-Index.
XMLoXSD	XML-Datentyp ohne XML Schema.
XMLoXSDPrim	XML-Datentyp ohne XML Schema aber mit primärem XML-Index.
XMLoXSDPrimSek	XML-Datentyp ohne XML Schema aber mit primärem und sekundärem PATH XML-Index.

Literaturverzeichnis

[AEN08] Yasser Abdel Kader, Barry Eaglestone, und Siobhan North. An Analysis of Relational Storage Strategies for Partially Structured XML. *the proceedings of the WebIST'08*, 2008. URL `http://staffwww.dcs.shef.ac.uk/people/S.North/papers/WebIST2008/webist2008paper.pdf`, (besucht am 03.02.2012).

[AG11] Software AG. Tamino XML Server Documentation, 2011. URL `http://documentation.softwareag.com/webmethods/tamino/ins82/overview.htm`, (besucht am 03.01.2012).

[B+02] Phil Bohannon et al. From XML Schema to relations: A Cost-Based Approach to XML Storage. 2002. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.112.7839`, (besucht am 01.02.2012).

[B+09] Denilson Barbosa et al. XML Storage. 2009. URL `http://hal.archives-ouvertes.fr/docs/00/43/34/34/PDF/Encyclopedia-XMLStorage.pdf`, (besucht am 20.01.2012).

[Bou05] Ronald Bourret. XML and Databases. 2005. URL `http://ece.ut.ac.ir/dbrg/seminars/AdvancedDB/2006/Sanamrad-Hoseininasab/Refrences/6.pdf`, (besucht am 19.01.2012).

[Col08] Michael Coles. *Pro SQL Server 2008 XML (Expert's Voice)*. Apress, 2008. ISBN 1-590-59983-7.

[FK99] Daniela Florescu und Donald Kossmann. Storing and Querying XML Data using an RDMBS, 1999. URL `http://classes.soe.ucsc.edu/cmps290s/Spring03/fk.pdf`, (besucht am 23.01.2012).

[J+01] Haifeng Jiang et al. Path Materialization Revisited : An Efficient Storage Model for XML Data. 2001. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.4074`, (besucht am 23.01.2012).

[K+05] Steffen Ulsø Knudsen et al. RelaXML : Bidirectional Transfer between Relational and XML Data. 2005. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.66.4376`, (besucht am 01.02.2012).

[KST02] Wassilios Kazakos, Andreas Schmidt, und Peter Tomczyk. *Datenbanken und XML: Konzepte, Anwendungen, System.* Springer-Verag Berlin Heidelberg, 2002. ISBN 3-540-41956-X.

[Max02] MaxiMedia. Mpower - Tamino integration. 2002.

[Max03] MaxiMedia. Mpower Agentur Spooler. 2003.

[Mic10] Microsoft. SQL Server 2008 R2 Books Online, 2010. URL http://msdn.microsoft.com/en-us/library/bb545450%28v=MSDN.10%29.aspx, (besucht am 05.01.2012).

[P+10] Marcus Paradies et al. Comparing XML processing performance in middleware and database: a case study. 2010. URL http://dl.acm.org/citation.cfm?id=1891725, (besucht am 16.02.2012).

[RTL03] RTL. Kurzreferenz zum Nachrichtenredaktionssystem „MPower“. 2003.

[S+99] Jayave Shanmugasundaram et al. Relational Databases for Querying XML Documents : Limitations and Opportunities. *SciencesNew York*, 1999. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.4288, (besucht am 23.01.2012).

[SCCSM10] Haw Su-Cheng, Lee Chien-Sing, und Norwati Mustapha. Bridging XML and Relational Databases: Mapping Choices and Performance Evaluation. *IETE Technical Review*, 2010. URL http://tr.ietejournals.org/article.asp?issn=0256-4602;year=2010;volume=27;issue=4;spage=308;epage=317;aulast=Su-Cheng, (besucht am 23.01.2012).

[Sch03] Harald Schöning. *XML und Datenbanken - Konzepte und Systeme*. Carl Hanser Verlag München Wien, 2003. ISBN 3-446-22008-9.

[Y+01] Masatoshi Yoshikawa et al. XRel : A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases. *Expert Systems*, 2001. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.3443, (besucht am 23.01.2012).

[ZO10] Ning Zhang und M.T. Özsu. XML Native Storage and Query Processing. *Advanced Applications and Structures in XML Processing: Label Streams, Semantics Utilization and Data Query Technologies. IGI Global*, 2010. URL http://www.igi-global.com/viewtitlesample.aspx?id=41497, (besucht am 03.02.2012).

Anhang

NCPower Querys an Tamino

Im Folgenden eine Auswahl von Querys zwischen *NCPower* und Tamino aus einem Trace auf einem Testsystem. Es handelt sich dabei nicht um alle Querys die mitgeschnitten wurden, da diese sich vom Aufbau her teilweise kaum unterscheiden. Das wird zum Beispiel bei den ersten drei Querys für die Admin-Collection deutlich. Alle drei haben einen Filter auf das gleiche Attribut, allerdings mit unterschiedlichen Werten.

Collection: Admin

```
/CompleteDocument[ BaseDocument/DocQualifier/DocReference[
 ( @uid='user/messner.xml' ) ]]
```

```
/CompleteDocument[ BaseDocument/DocQualifier/DocReference[
 ( @uid='rolegroup/cvd_kein_TP.xml' ) or
 ( @uid='usergroup/edv.xml' ) ]]
```

```
/CompleteDocument[ BaseDocument/DocQualifier/DocReference[
 ( @uid='keystrokestate/messner.xml' ) or
 ( @uid='menu/admin.xml' ) or
 ( @uid='navigation/default.xml' ) or
 ( @uid='windowstate/messner.xml' ) ]]
```

```
/CompleteDocument[
 (BaseDocument/DocUserFields/Type='USER') ]
 sortby( @ino:id desc )/BaseDocument
```

Collection: Messaging

```
/CompleteDocument[
 (DocContent/Document/Message/OWNER~='messner') and
 (DocContent/Document/Message/STATUS~='new') and
 (BaseDocument/DocUserFields/Type='Message') ]
 sortby( @ino:id asc )/BaseDocument
```

Collection: News

```
/CompleteDocument[
 ( (DocContent/Document/NITF/AGENCY~='SID') or
 (DocContent/Document/NITF/AGENCY~='AFP') or
 (DocContent/Document/NITF/AGENCY~='Reuters') or
 (DocContent/Document/NITF/AGENCY~='DPA B') or
 (DocContent/Document/NITF/AGENCY~='SATWAS') or
 (DocContent/Document/NITF/AGENCY~='AP'))  ]
 sortby( @ino:id desc )/BaseDocument
```

```
/CompleteDocument[
 ( (DocContent/Document/NITF/AGENCY~='SID') or
 (DocContent/Document/NITF/AGENCY~='AFP') or
 (DocContent/Document/NITF/AGENCY~='Reuters') or
 (DocContent/Document/NITF/AGENCY~='DPA B') or
 (DocContent/Document/NITF/AGENCY~='SATWAS') or
 (DocContent/Document/NITF/AGENCY~='AP'))  and
 (DocContent/Document/NITF/CATEGORY~='KULTUR') ]
 sortby( @ino:id desc )/BaseDocument
```

```
/CompleteDocument[ ( (DocContent/Document~='Sport') and
 (DocContent/Document~='Fußball'))  and
 ( (DocContent/Document/NITF/AGENCY~='SID') or
 (DocContent/Document/NITF/AGENCY~='AFP') or
 (DocContent/Document/NITF/AGENCY~='Reuters') or
 (DocContent/Document/NITF/AGENCY~='DPA B') or
 (DocContent/Document/NITF/AGENCY~='SATWAS') or
 (DocContent/Document/NITF/AGENCY~='AP'))  ]
 sortby( @ino:id desc )/BaseDocument
```

```
/CompleteDocument[ (DocContent/Document~='Sport') and
 ( (DocContent/Document/NITF/AGENCY~='SID') or
 (DocContent/Document/NITF/AGENCY~='AFP') or
 (DocContent/Document/NITF/AGENCY~='Reuters') or
 (DocContent/Document/NITF/AGENCY~='DPA B') or
 (DocContent/Document/NITF/AGENCY~='SATWAS') or
 (DocContent/Document/NITF/AGENCY~='AP'))  ]
 sortby( @ino:id desc )/BaseDocument
```

```
/CompleteDocument[ ( (DocContent/Document~='Sport') and  not
 (DocContent/Document~='Fußball'))  and
 ( (DocContent/Document/NITF/AGENCY~='SID') or
 (DocContent/Document/NITF/AGENCY~='AFP') or
 (DocContent/Document/NITF/AGENCY~='Reuters') or
 (DocContent/Document/NITF/AGENCY~='DPA B') or
```

```
 (DocContent/Document/NITF/AGENCY~='SATWAS') or
 (DocContent/Document/NITF/AGENCY~='AP'))  ]
 sortby( @ino:id desc )/BaseDocument
```

```
count( /CompleteDocument[ (DocContent/Document~='Sport') and
 ( (DocContent/Document/NITF/AGENCY~='SID') or
 (DocContent/Document/NITF/AGENCY~='AFP') or
 (DocContent/Document/NITF/AGENCY~='Reuters') or
 (DocContent/Document/NITF/AGENCY~='DPA B') or
 (DocContent/Document/NITF/AGENCY~='SATWAS') or
 (DocContent/Document/NITF/AGENCY~='AP'))  ]/BaseDocument )
```

```
/CompleteDocument[ ( (DocContent/Document~='Sport') and  not
 (DocContent/Document~='Fuball'))  and
 ( (DocContent/Document/NITF/AGENCY~='SID') or
 (DocContent/Document/NITF/AGENCY~='AFP') or
 (DocContent/Document/NITF/AGENCY~='Reuters') or
 (DocContent/Document/NITF/AGENCY~='DPA B') or
 (DocContent/Document/NITF/AGENCY~='SATWAS') or
 (DocContent/Document/NITF/AGENCY~='AP'))  and
 (( @ino:id > 6268382)) ] sortby( @ino:id asc )/BaseDocument
```

```
/CompleteDocument[
 ( (DocContent/Document/NITF/AGENCY~='SID') or
 (DocContent/Document/NITF/AGENCY~='AFP') or
 (DocContent/Document/NITF/AGENCY~='Reuters') or
 (DocContent/Document/NITF/AGENCY~='DPA B') or
 (DocContent/Document/NITF/AGENCY~='SATWAS') or
 (DocContent/Document/NITF/AGENCY~='AP'))  and
 (( @ino:id[1] < 6268381)) ]
 sortby( @ino:id desc )/BaseDocument
```

```
/CompleteDocument[
 ( (DocContent/Document/NITF/AGENCY~='BILD'))  ]
 sortby( @ino:id desc )/BaseDocument
```

```
/CompleteDocument[
 BaseDocument/DocSysValues/CreationDate>1313532000 and
 BaseDocument/DocSysValues/CreationDate<1313586320 and
 ( (DocContent/Document/NITF/AGENCY~='SID') or
 (DocContent/Document/NITF/AGENCY~='AFP') or
 (DocContent/Document/NITF/AGENCY~='Reuters') or
 (DocContent/Document/NITF/AGENCY~='DPA B') or
 (DocContent/Document/NITF/AGENCY~='SATWAS') or
 (DocContent/Document/NITF/AGENCY~='AP'))  ]
 sortby( @ino:id desc )/BaseDocument
```

Collection: CMS

```
/CompleteDocument[ (BaseDocument/DocUserFields/Type='VOS')
and ( (DocContent/Document/VOS/VOSSTATUS~='OK') or
(DocContent/Document/VOS/VOSSTATUS~='FUSED') or
(DocContent/Document/VOS/VOSSTATUS~='NOLINK') or
(DocContent/Document/VOS/VOSSTATUS~='RECORDING') or
(DocContent/Document/VOS/VOSSTATUS~='COMPLETE'))  and
( (DocContent/Document/VOS/RUNDOWN='') or
(DocContent/Document/VOS/RUNDOWN~='rdformats/demo.xml'))
and (  not (DocContent/Document/VOS/OBJGROUP~='RTLCOLOGNE')
and  not (DocContent/Document/VOS/OBJGROUP~='INCOLOGNEHD')
and  not (DocContent/Document/VOS/OBJGROUP~='RTLHAMBURG')
and  not (DocContent/Document/VOS/OBJGROUP~='RTLFRANKFURT')
and  not (DocContent/Document/VOS/OBJGROUP~='RTLMUNICH')
and  not (DocContent/Document/VOS/OBJGROUP~='RTLBERLIN')
and  not (DocContent/Document/VOS/OBJGROUP~='VPMSTEST')
and  not (DocContent/Document/VOS/OBJGROUP~='NTVCOLOGNE')
and  not (DocContent/Document/VOS/OBJGROUP~='NTVBERLIN')
and  not (DocContent/Document/VOS/OBJGROUP~='RTLWEST')
and  not (DocContent/Document/VOS/OBJGROUP~='RTL2NEWS'))
and((DocContent/Document/VOS/SHOTS/objSubGroup/objMember='')
or
(DocContent/Document/VOS/SHOTS/
      objSubGroup/objMember~='default')
or (DocContent/Document/VOS/SHOTS/
      objSubGroup/objMember~='edv')
or
(DocContent/Document/VOS/SHOTS/
      objSubGroup/objMember~='default')) ]
sortby( @ino:id desc )/BaseDocument
```

```
/CompleteDocument[ BaseDocument/DocQualifier/DocReference[
 ( @uid='62296543_0' ) ]]
```

Collection: Editorials

```
/CompleteDocument[
(BaseDocument/DocUserFields/Type='RUNDOWN') and
(DocContent/Document/
     RUNDOWN/RDFORMAT~='rdformats/demo.xml')]
```

```
sortby( DocContent/Document/
     RUNDOWN/HEADER/RUNDOWNDATE desc )/BaseDocument
```

```
/CompleteDocument[
BaseDocument/DocQualifier/DocReference[  @version='1'] and
BaseDocument/DocQualifier/DocConfigName
   [  @configName='RD Demo 2011/08/18 14:00:00' and
   @configNumber='1' ]]
sortby(@ino:id desc )/BaseDocument/DocQualifier
```

```
/CompleteDocument[ BaseDocument/DocQualifier/DocReference[
 ( @uid='MMT-4-RUNDOWN-1313419433695-253061' ) ]]
```

```
/CompleteDocument[
(DocContent/Document/RUNDOWN/HEADER/
     RUNDOWNTIME>='2011/08/18 00:00:00') and
(DocContent/Document/RUNDOWN/HEADER/
     RUNDOWNTIME<='2011/08/18 23:59:59') and
(DocContent/Document/RUNDOWN/RDFORMAT~='rdformats/demo.xml')
and (BaseDocument/DocUserFields/Type='RUNDOWN') ]
sortby( @ino:id desc )/BaseDocument
```

Collection: Beliebig

```
/CompleteDocument[ @ino:id=6268382]
```

Ergebnisse der Performance-Tests

Struktur	Zeit	Größe	Indexgröße (Prim)	Indexgröße (Prim/Sek)
Original	-	552.320 KB	-	-
XMLoXSD	908,69 s	1.068.114 KB (+ 81.216 KB Index)	1.800.920 KB	2.293.432 KB
XMLmXSD	1047,95 s	1.166.352 KB (+ 56.256 KB Index)	1.635.976 KB	2.033.088 KB
h1	3963,74 s	Base: 753.880 KB Content: 596.184 KB	-	Base: 1.179.872 KB Content: 902.632 KB
h2	925,16 s (+ 908,69 s)	Base: 623.552 KB Content: 596.184 KB	-	Base: 796.368 KB Content: 902.632 KB
rel	666,35 s (+ 908,69 s)	265.320 KB (+ 79.856 KB Index)	-	-

Tabelle A1 - Speicherplatzverbrauch und Beladungszeit der Strukturen

Query	XMLoXSD	XMLoXSDPrim	XMLoXSDPrimSek
Q1.1k	0,0547	0,1807	0,0900
Q1.1t	0,0460	0,0567	0,0300
Q1.2k	3,3727	2,5453	2,1900
Q1.2t	1,1027	1,0197	1,2687
Q1.2s	1,6490	1,6980	1,8000
Q2.1k	175,3280	63,3050	81,5500
Q2.1t	82,2987	69,7633	69,7900
Q2.1s	66,2720	71,9783	73,4100
Q2.2k	126,2097	217,1840	226,9480
Q2.2t	98,2253	261,0280	214,5727
Q2.2s	221,1590	249,4683	215,7760
Q2.3k	376,4597	263,0010	213,8400
Q2.3t	225,5257	221,9030	187,0323
Q2.4k	66,6850	124,4173	83,9920
Q2.4t	56,4697	96,3527	79,6420
Q2.5k	23,8120	35,8543	0,2817
Q2.5t	15,5623	46,5660	0,1683
Q2.6k	16,0090	36,9377	0,2657
Q3.1k	57,7033	72,7110	70,2193
Q3.1t	55,6980	48,2973	69,9093
Q3.1s	55,0277	56,1770	80,1430
Q3.2k	194,9847	184,5193	184,8153
Q3.2t	168,2303	163,5417	136,7213
Q3.2s	183,4820	187,0990	166,7657
Q3.3k	19,4040	112,9947	94,3877
U1	1,9617	0,1457	0,1123
I1	0,6620	0,0643	0,0817
D1	0,2930	0,0520	0,0520
D2	0,0630	0,0990	0,1550

Tabelle A2 - Durchschnittliche Laufzeiten (in s) der Querys für die XML-Datentyp Strukturen ohne XSD aber mit verschiedenen Indizes

Query	XMLmXSD	XMLmXSD-Prim	XMLmXSD-PrimSek	XMLmXSD-PrimSek-PROP
Q1.1k	0,4357	0,1187	0,0373	-
Q1.1t	0,0413	0,0530	0,0063	-
Q1.2k	2,9720	1,3087	1,3067	-
Q1.2t	1,7353	1,1267	1,1523	-
Q1.2s	1,6020	1,3717	1,2063	-
Q2.1k	63,4653	26,9180	1,5987	-
Q2.1t	51,3630	30,6653	0,9357	-
Q2.1s	43,9493	24,0787	1,4423	-
Q2.2k	120,0200	297,5423	275,5977	322,8447
Q2.2t	83,2203	244,2953	196,7290	-
Q2.2s	107,6030	220,1760	267,4260	-
Q2.3k	137,9323	317,2930	260,1350	-
Q2.3t	101,2513	152,0130	234,2990	-
Q2.4k	34,7380	32,5710	1,7220	-
Q2.4t	34,0577	29,5667	1,2730	-
Q2.5k	24,0980	29,1920	0,0713	-
Q2.5t	24,5997	29,2920	0,0593	-
Q2.6k	15,7087	28,4903	0,0177	-
Q3.1k	35,8913	30,5800	1,4657	-
Q3.1t	34,4903	30,4023	1,3070	-
Q3.1s	35,6967	23,6620	1,2400	-
Q3.2k	135,8440	185,4613	206,7690	258,9230
Q3.2t	106,9520	100,9097	182,5030	-
Q3.2s	125,2537	122,6530	154,8487	-
Q3.3k	16,5287	16,5600	0,2323	-
U1	0,0753	0,1143	0,0497	-
I1	0,0600	0,0673	0,1260	-
D1	0,1910	0,3760	0,0550	-
D2	0,1270	0,1330	0,1250	-

Tabelle A3 - Durchschnittliche Laufzeiten (in s) der Querys für die XML-Datentyp Strukturen mit XSD und mit verschiedenen Indizes

Query	h1	h2	rel
Q1.1k	0,5027	0,1697	0,0443
Q1.1t	0,0700	0,0227	0,0013
Q1.2k	1,6123	0,9017	0,2530
Q1.2t	0,9410	0,1893	0,2903
Q1.2s	1,5143	0,7603	0,3990
Q2.1k	2,8883	19,6877	0,4863
Q2.1t	1,1647	0,3633	0,4387
Q2.1s	1,5657	0,8560	0,4627
Q2.2k	470,8847	506,2867	37,3573
Q2.2t	266,3017	196,2387	9,9037
Q2.2s	690,7010	404,6680	19,8573
Q2.3k	375,8117	280,3060	16,8697
Q2.3t	121,5950	88,9530	9,8987
Q2.4k	2,2347	9,7903	0,7667
Q2.4t	1,3033	1,4317	0,5463
Q2.5k	0,2193	7,7950	0,2117
Q2.5t	0,2037	2,7097	0,1513
Q2.6k	0,2200	0,5417	0,2107
Q3.1k	4,9847	5,6867	0,7393
Q3.1t	4,6700	4,9910	0,5767
Q3.1s	5,1133	4,7010	0,6863
Q3.2k	554,6290	380,5907	18,5803
Q3.2t	425,1127	316,4543	10,0423
Q3.2s	574,7180	413,5423	18,4780
Q3.3k	4,0230	6,7750	0,4900
U1	0,0820	0,1310	0,0577
I1	0,1067	0,1313	0,1690
D1	0,0570	0,1290	0,0370
D2	0,1090	0,2640	0,0430
Z1	-	-	0,8900
Z2	-	-	4,0523

Tabelle A4 - Durchschnittliche Laufzeiten (in s) der Querys für die relationale und die hybriden Strukturen

	TBI	TIB	Volltext	TOP 10 XMLmXSD
Q2.2k	212,8137	123,7777	-	-
1 Wort	-	-	1,1093	-
6 Wörter	-	-	1,2063	-
1 Wort + XML-Dokument	-	-	153,5510	-
Q3.2k	-	-	-	1,0963

Tabelle A5 - Durchschnittliche Laufzeiten (in s) für die Zusatz-Tests: Indexreihenfolge (TBI, TIB), Volltextsuche über XML (Volltext), TOP 10 für die Struktur XMLmXSD

Inhalt der CD

Bei dieser Arbeit wurden Test-Querys für verschiedene Strukturen erstellt. Um der Semantik von *NCPower*-Querys an Tamino zu entsprechen, sind diese teilweise recht komplex. Aufgrund des Umfangs dieser Daten und der zugehörigen Messergebnisse, befinden sich diese auf der beiligenden CD. Zusätzliche wurden die Ergebnisse der dieser Arbeit vorangegangenen Untersuchungen ebenfalls beigelegt. Die CD hat folgenden Inhalt:

- Test-Querys für die einzelnen Strukturen
- Logs der Messergebnisse
- SQL Statements zum Beladen der Strukturen
- Praxisprojekt Dokumentation „Untersuchung der Übersetzbarkeit zweier XQuery-Dialekte und Entwicklung eines Übersetzers“
- Tamino-Gateway Prototyp

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbständig verfasst zu haben.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben.

Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsgbehörde vorgelegen.

Gummersbach, 05. März 2012

Christopher Messner